The Application of Neural Operators to Predict Skin Evolution After Burn Trauma

MSc Thesis

Selma Husanovic

Delft University of Technology

TU Delft

The Application of Neural Operators to Predict Skin Evolution After Burn Trauma

MSc Thesis

by

Selma Husanovic

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Tuesday September 3, 2024 at 1:00 PM.

Student number:	4587014	
Project duration:	July, 2023 – September, 2024	
Thesis committee:	Dr. A. Heinlein,	TU Delft, daily supervisor
	Prof. dr. F. J. Vermolen	UHasselt, supervisor
	Prof. dr. ir. M.B. van Gijzen,	TU Delft
	Dr. ir. E.G. Rens,	TU Delft

TU Delft

Abstract

Burn injuries present a significant global health challenge. Among the most severe long-term consequences are *contractures*, which can lead to functional impairments and disfigurement. Understanding and predicting the evolution of post-burn wounds is crucial for developing effective treatment strategies. Traditional mathematical models, while accurate, are often computationally expensive and time-consuming, limiting their practical application. Recent advancements in machine learning, particularly in deep learning, offer promising alternatives for accelerating these predictions. This study investigates the use of a *deep operator network* (DeepONet), a type of neural operator, as a surrogate model for finite element simulations for predicting post-burn wound evolution. We trained DeepONets on various wound shapes, enhancing the architecture by incorporating initial wound shape information and applying sine augmentation to enforce boundary conditions. The most sophisticated model achieved an R^2 score of 0.9960, indicating strong predictive accuracy. Additionally, the model generalised well to convex combinations of basic shapes, with an R^2 score of 0.9944, and provided reliable predictions over an extended period of up to one year. These findings suggest that DeepONets can effectively serve as a surrogate for traditional finite element methods in simulating post-burn wound evolution, with potential applications in medical treatment planning.

Acknowledgements

This thesis marks the conclusion of my long journey at the TU Delft. I would not have made it this far without the support and guidance of several special individuals. Firstly, I would like to express my deepest gratitude to my daily supervisor, Alexander Heinlein. Your exceptional supervision, marked by your attentiveness, promptness, enthusiasm, and your wealth of ideas, has been invaluable to me. I also wish to extend my sincere appreciation to my co-supervisor, Fred Vermolen. Your valuable input and ideas, your most contagious enthusiasm for my results, and encouraging words have greatly enriched my experience. A special thanks goes out to Ginger Egberts. Thank you for providing me with the finite element code that made this entire work possible and for the time you took to extend it, such that it was more tailored to my ideas. I am also grateful for your prompt communication and readiness to offer thoughtful ideas, whenever I was in need of help. Furthermore, I would like to thank Martin van Gijzen en Lisanne Rens for being part of my thesis committee and for your valuable input during my literature presentation. Lastly, my heartfelt thanks go out to my friends, family, and loved ones. Your unwavering support made this journey possible, and for that, I am forever grateful.

Selma Husanovic
Delft, August 2024

Contents

1 Introduction

Burn injuries are a global public health issue, responsible for an estimated 180.000 annual deaths, according to the World Health Organization [98]. Beyond the immediate threat to life, non-fatal burns are a leading cause of morbidity, presenting significant challenges for individuals, both physically and psychologically. They can lead to prolonged hospitalisation, disfigurement, and disability, which can result in social stigma and rejection [66]. Functional impairments and immobility can occur when a post-burn wound is very large or in close proximity to joints. This is due to a biomechanical process called *contraction*: the edges of the wound draw together, reducing its size and resulting in a deformation. This is a natural bodily response aiding in the healing process, however, when it becomes excessive and long-term, we speak of a *contracture*. Contractures, along with *hypertrophic scarring*, characterised by stiff, raised, and uneven-textured scars, are among the most challenging long-term effects of severe burns.

Research into burn injury treatment is a highly active field, with developments in wound dressing [78], excision techniques [90], skin grafting [96], hyper-metabolism [86], pain management [80], and many more. Understanding the evolution of wounds following burn trauma is very important for clinicians to devise effective treatment strategies. Various disciplines study the prevention of contractures, including medical sciences, biology, and mathematics. Mathematical modelling offers a powerful tool for gaining insights into the complex biological and physical processes that govern post-burn wound healing and contraction. By simulating these processes, models can predict wound behaviour over time, identify important factors influencing contracture formation, and guide the development of more effective treatment strategies.

Over the years, various mathematical models have been developed to predict the behaviour of wound healing and contraction. They can be subdivided into three categories: *agent-based* models that simulate individual cells [4, 94], *continuum-based* models that simulate the entire tissue using partial differential equations (PDEs) [41, 45], and *cellular automata* that simulate tissue using spatial Markov chains [95]. For a more extensive overview, we refer the reader to [93]. While numerical models have proven effective in providing accurate approximations of wound evolution, they come with limitations. They are often computationally expensive, requiring substantial time for execution, and altering parameters necessitates rerunning the entire model. This can limit their practical application, especially when timely results are crucial [67].

Recent research in the medical field has shifted towards exploring the potential of machine learning, specifically *deep neural networks*. These networks have been applied in many areas, such as medical imaging [42], decease recognition [65, 85], and predictive diagnostics [2]. One of the advantages of using neural networks is that they can be trained to learn complex relationships within a relatively short evaluation time. As a result, there is an increased interest in the combination of both deep learning and mathematical modelling, to combine the strength of both. One application is the use of neural networks as *surrogate models*, to approximate and replace computationally costly numerical simulations. Examples can be found in cardiovascular applications [49, 51, 52, 56] and soft tissue modelling of organs

[62, 71, 73].

The exploration of neural network surrogates for modelling post-burn wound evolution has gained recent attention as well. Egberts et al. use simple, feedforward neural networks to replace expensive finite-element simulations that predict post-burn contraction and patient discomfort for a one-dimensional [20] and two-dimensional [18] wound. Their work demonstrates the effectiveness of neural networks in reproducing the finite element results, offering a cost-effective and significantly faster alternative. A limitation of their work is that all simulations are performed on a fixed spatial domain, meaning the wound's size and shape remain constant. Egberts et al. acknowledge that one of the directions for further research is investigating neural networks capable of incorporating varying wound shapes. A logical first step would be to use standard geometrical shapes for the wound, such as circles and squares. Ultimately, the goal would be to simulate wounds of any shape. One approach the authors suggest is using a *convolutional neural network* that processes an image of the initial wound.

One class of deep learning architectures that has the primary application of learning surrogate maps for the solution operators of PDEs are *neural operators* [48]. In recent years, several neural operator architectures have emerged, with *Fourier neural operators* (FNOs) [50] and *deep operator networks* (DeepONets) [54] being the most widely used. They are designed to learn mappings between infinite-dimensional function spaces. Neural operators have demonstrated improved performance in solving PDEs compared to existing machine learning methodologies, while being significantly faster than numerical solvers [50]. DeepONets and its extensions have shown strong performance in diverse applications, including multi-scale and multi-physics problems [6], fluid dynamics [12, 58], power grids [63], and solar-thermal systems [69]. Some applications in biology and medicine include bubble growth dynamics [53] and aortic dissection [100].

In this study, we propose the use of neural operators as a surrogate model to reproduce finite-element simulations that predict post-burn wound evolution over time. Specifically, we will focus on DeepONets, for learning operators accurately and efficiently from a relatively small dataset [54]. Our object is to learn the solution operator of the two-dimensional *morphoelastic model*, which describes post-burn wound evolution, thus accurately reproducing the finite element predictions. A key aspect of this research will be to incorporate multiple initial wound shapes, as this has never been done before. In this way, our research aims to extend the works of Egberts et al. We formulate the following main research question:

Can we train a neural operator based on the DeepONet architecture to accurately predict post-burn wound evolution over time, while accounting for multiple initial wound shapes?

To help answer this question, we formulate three sub-questions:

1. Can we train a DeepONet on one specific wound shape, to accurately predict the dermal displacement across the entire domain over time?
2. Can we train a DeepONet on *multiple wound shapes*, to accurately predict the dermal displacement across the entire domain over time?
3. Can we train a DeepONet on multiple wound shapes, to accurately predict the dermal displacement across the entire domain over time, that *generalises well to convex combinations of the basic shapes*?

The study is organised in the following way: Chapter 2 provides a foundational understanding of the biology of dermal wound healing, setting the stage for subsequent discussions. Chapter 3 presents the morphoelastic mathematical model for burn injuries, connecting biological principles to mathematical formulations. Chapter 4 gives a comprehensive overview of neural networks and explains the concept of neural operators, specifically DeepONets, which we apply to predict post-burn wound evolution. Our three sub-questions are addressed in Chapters 5 to 7. Each chapter focuses on predictions for single wound shapes, multiple wound shapes, and convex combinations of wound shapes, respectively. The conclusion of our study is formulated in Chapter 8, where we answer our main research question. Lastly, Chapter 9 discusses the limitations of our work and gives recommendations for future research.

2

The Biology of Dermal Wound Healing

This chapters serves to give a comprehensive description of the current biological understanding of dermal wound healing. Section 2.1 gives a brief introduction into skin anatomy and physiology. Section 2.2 highlights the *extracellular matrix*, a vital component of the skin that needs to be rebuilt during wound healing. Section 2.3 delves into the four phases of wound healing and explains the biological and mechanical processes at work. Lastly, Section 2.4 focuses on skin injuries due to severe burns. The emphasis is on complications due to an abnormal healing response.

2.1. The Human Skin

The human skin is a complex, multilayered organ, consisting of heterogeneous cell types and extracellular components [23]. It is one of the largest organs in the human body, having a surface area of approximately $2m^2$ and making up 16% of the total body weight [72]. Amongst its many functions, the most important ones are preventing the organism from dehydrating, while protecting it from its environment. Skin is dynamic, able to heal itself and responsive to the external environment, ensuring human survival [24]. As schematically depicted in Figure 2.1, the skin consists of three layers: the *epidermis*, the *dermis* and the *hypodermis*.

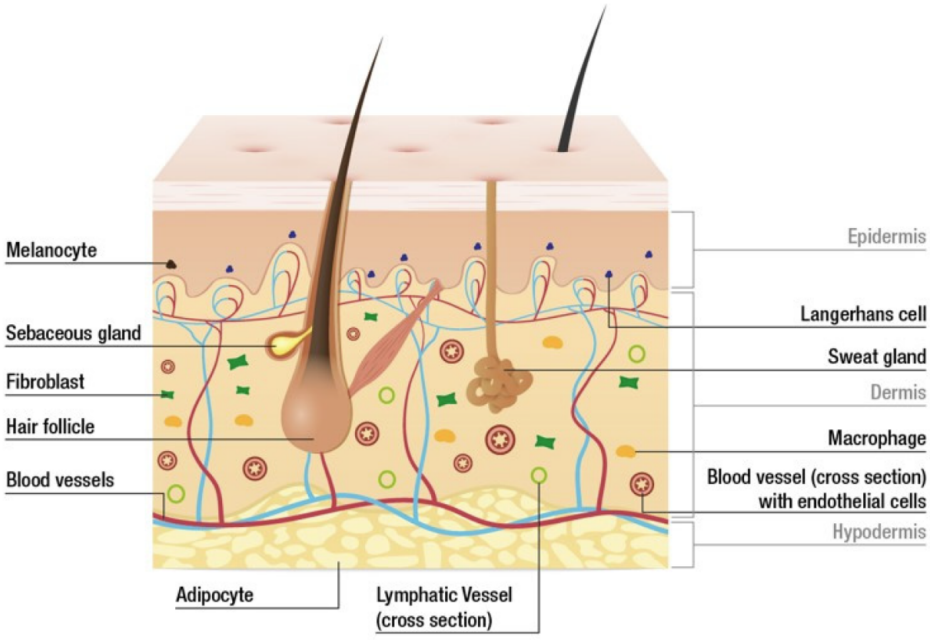


Figure 2.1: Schematic structure of human skin. Taken from [39].

The epidermis is the outermost layer and is approximately 0.1mm thick, although thickness can vary depending on the location [24]. It primarily functions as a protective barrier. The main cell type in the epidermis is the *keratinocyte*, which constitutes 90% of epidermal cells [60]. Keratinocytes differentiate upwards through the epidermis. Their maturation stages can be divided into four physical layers. It takes approximately 35 days for the whole epidermal layer to be replaced by new cells [72].

Immediately below and connected to the epidermis lies the dermis. Its thickness varies between 0.3mm (e.g., on the eyelids) and 0.6mm (e.g., on the back and soles) [61]. The dermis serves numerous valuable purposes: it provides firmness, flexibility and tensile strength to the skin. Moreover, it binds water, regulates the temperature and contains receptors of sensory stimuli [23]. The dermis is less cellular than the epidermis, consisting primarily of fibrous extracellular matrix (ECM), that surrounds the dermal cells and other constituents (e.g. neurovascular network, sensory receptors). The ECM plays an important role in wound healing and will be treated in more detail in the subsequent section. The most abundant cell type in the dermis is the *fibroblast*, that migrates through the tissue and is responsible for producing and maintaining the ECM [24]. Other skin cells present are immune cells that protect from pathogens, e.g., *macrophages*; mast cells that are involved in allergic reactions, blood vessel, and nerve cells [61].

Directly below the dermis lies the hypodermis (sometimes called the subcutis), a layer of loose connective tissue and fat. Subcutaneous tissue varies in thickness across the body, additionally depending on the sex of the individual [72]. It insulates the body, serves as a reserve energy supply and cushions the skin [23].

2.2. The Extracellular Matrix

The extracellular matrix is often referred to as the “ground substance” of the dermis. It is an intricate structure of different molecules that fills the space between skin cells and provides structural integrity, elasticity and mechanical strength to the tissue. Figure 2.2 provides a schematic visualisation of the ECM in healthy human skin. The following subsections aim to give a detailed overview of ECM anatomy and physiology (Section 2.2.1), with special emphasis on ECM remodelling (Section 2.2.2).

2.2.1. Composition of the ECM

The major constituent of the ECM is *collagen*, a protein making up 70% of the dermis [24]. The periodically banded collagen fibers form a network that provides tensile strength and structural support [74]. Different types of collagen are present, the most abundant ones being type I, III, and V [61].

Another central component of the ECM is *elastin*, a protein that provides elasticity to the skin. Elastic fibres return the skin to its normal configuration after being stretched or deformed [23]. Elastin and collagen together maintain skin’s firmness and flexibility.

Additionally, the ECM contains *proteoglycans* (PGs) and *glycosaminoglycans* (GAGs). These are large molecules that are interspersed between and stick to collagen fibers in the ECM [74]. An example of a GAG is hyaluronic acid. PGs and GAGs are able to bind water molecules, providing a gel-like milieu and thus regulating the hydration of the skin. They effectively ensure skin’s plumpness and smoothness [61].

Glycoproteins like *fibronectin* and *laminin* are adhesive molecules that promote cell attachment to the ECM. One of the functions of the ECM is to act as a scaffold for the skin cells, enabling them to adhere and migrate through the matrix. Fibronectin and laminin provide these anchor points.

The ECM is a reservoir for *signalling molecules*, such as *growth factors*, *cytokines*, and *hormones*, that are involved in cell communication. Skin cells have receptors on their surface that are able to recognise specific signalling molecules. When one of the latter binds, it triggers a certain response from the cell (e.g., migration or the secretion of a specific molecule) [5]. Growth factors, a specific kind of signalling molecules, regulate the growth, proliferation, and differentiation of cells [5].

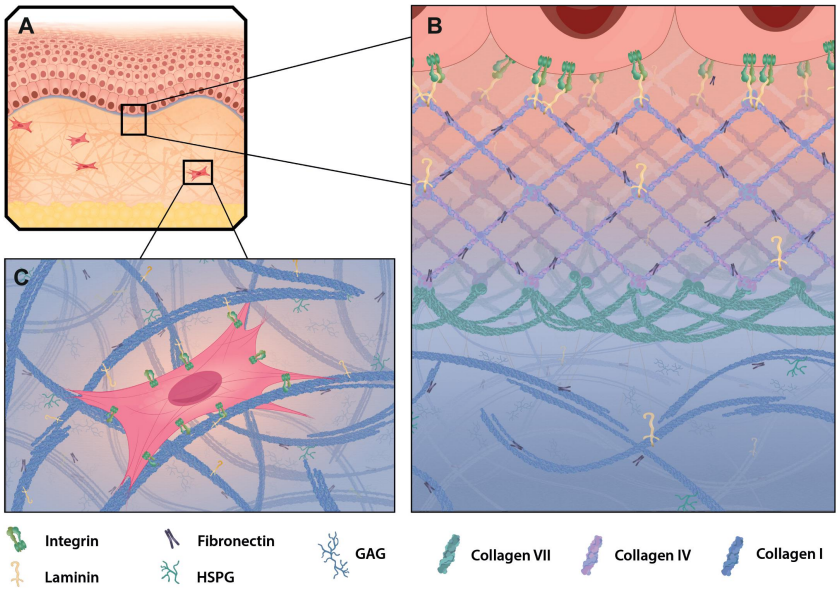


Figure 2.2: ECM in healthy human skin. The three skin layers are clearly visible (A). The ECM is a complex network of molecules that surrounds and supports the cells (B). It has attachment sites for the skin cells, allowing for cell adhesion and migration through the matrix (C). Taken from [74].

2.2.2. ECM Remodelling

The ECM is a highly dynamic structure that is constantly being remodelled. This entails a process in which ECM components are synthesised, deposited, degraded, and reorganised [55]. *Fibroblasts* and *myofibroblasts* are key cellular players in ECM remodelling.

As discussed in Section 2.1, fibroblasts are the most common type of cells in the dermis. They are responsible for synthesising and depositing the components that constitute the ECM. Fibroblasts can differentiate into myofibroblasts, resulting into a specialised skin cell that has contractile properties, resembling both fibroblasts and smooth muscle cells [38]. Myofibroblasts are also involved in producing constituents of the ECM.

An important instance in which ECM remodelling occurs is during wound healing and tissue repair. In the case of a dermal wound, part of the ECM is completely destroyed and requires reconstruction [22]. Here, fibroblast-to-myofibroblast differentiation occurs. Myofibroblasts play an important role during wound healing: through their contractile properties, they exert mechanical forces on the ECM, contributing to its realignment and restructuring [38]. This additionally leads to wound *contraction*: the edges of the wound draw together, facilitating the healing process [38]. The subsequent section will delve more into the complex process of wound healing.

2.3. Phases of Wound Healing

Wounds to the skin can be categorised in a number of different ways. One distinction is between epidermal and dermal wounds. In this work, the focus is on the latter, where we assume that the wound is as deep as to affect the dermis, where the ECM is damaged or destroyed.

Dermal wound healing is a complex sequence of overlapping events which are often described separately, but in reality form a continuum referred to as *the healing cascade* [14]. For the ease of explanation, we shall also make the distinction of four separate phases: *hemostasis*, *inflammation*, *proliferation*, and *remodelling*. Sections 2.3.1 to 2.3.4 serve to explain the current biological understanding of each healing phase in more detail. Please refer to Figure 2.3 for a schematic overview.

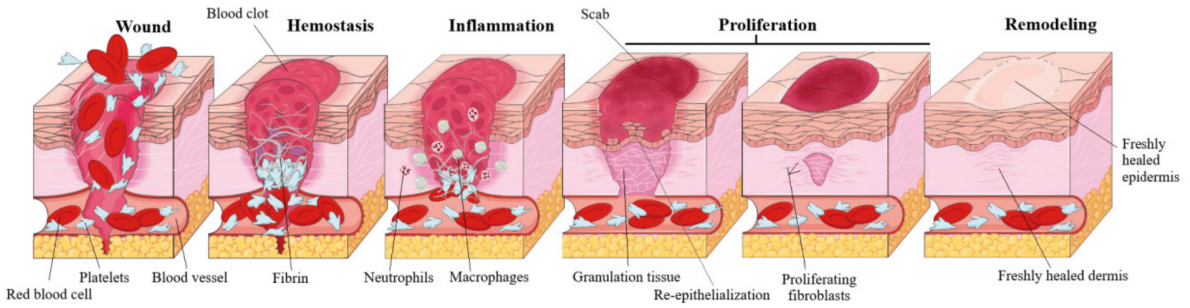


Figure 2.3: Healing cascade phases of dermal wounds. Taken from [26].

2.3.1. Hemostasis

The moment healthy tissue is injured, the healing response commences. Blood vessels in the area constrict to reduce blood flow [25]. As the blood components spill into the site of the injury, the platelets come into contact with exposed collagen and other elements of the ECM [14]. This triggers the platelets to release clotting factors. They aggregate at the site of injury, forming a blood clot to stop the bleeding. The fibrin clot acts as a temporary barrier and releases signalling molecules to initiate healing [9]. It also serves as a temporary matrix through which cells can migrate [25].

2.3.2. Inflammation

Hemostasis is followed by the inflammatory phase, which is characterised by the removal of pathogens. Now, the blood vessels near the injury dilate and become more permeable. This promotes the transportation of leukocytes (*neutrophils* and *monocytes*), which adhere to the blood vessel walls and migrate out of the bloodstream, into the tissue [32].

Neutrophils are the first to arrive at the site of injury to combat pathogens. The major function of the neutrophil is to remove foreign material, bacteria, non-functional host cells, and damaged matrix components that may be present in the wound site [14]. Neutrophils have a relatively short lifespan.

Monocytes differentiate into *macrophages*, which are very potent immune cells that engulf and kill pathogens [60]. They are also responsible for removing nonfunctional host cells, bacteria-filled neutrophils, damaged ECM, foreign debris, and any remaining bacteria from the wound site [14].

Macrophages release certain signalling molecules, such as growth factors and cytokines, to recruit even more immune cells to the site of injury [25]. The inflammatory response causes redness, swelling and wamth in the wounded area. The presence of wound macrophages is a marker that the inflammatory phase is nearing an end and that the proliferative phase is beginning [14].

2.3.3. Proliferation

Once the wound site is cleaned out, fibroblasts from the surrounding undamaged skin migrate in and proliferate. They synthesise and deposit new ECM components, including collagen, to rebuild the tissue framework [25].

Another predominant cell type proliferating during this phase is the *endothelial cell* (i.e., cells that form the walls of blood vessels). New blood vessels are formed through a process called *angiogenesis*, supplying oxygen and nutrients to support healing [14].

Initially, granulation tissue is formed, consisting of proliferating fibroblasts, newly formed blood vessels and loose ECM. This is temporary tissue that is later substituted for the real ECM [74]. The granulation tissue fills the wound from the base up, gradually filling the wound defect [22].

Signalling molecules induce fibroblasts already located in the wound site to transform into myofibroblasts [25], which exhibit less proliferation compared to the fibroblasts migrating from the wound periphery. Myofibroblasts are additionally responsible for producing constituents of the ECM [38]. As myofibroblasts are able to exert large contractile forces, their presence effectively turns the granulation tissue into a temporary contractile organ [57]. This pulls the wound edges toward the center, which results in a gradual reduction of the wound area [47].

During the proliferative phase, *epithelisation* occurs: epithelial cells (keratinocytes) from the wound edges start to migrate and proliferate across the wound periphery. They gradually cover the granulation tissue and in doing so effectively close the wound [47].

2.3.4. Remodelling

The remodelling phase is the longest healing phase and can take up to a year [47]. Clinically, this is perhaps the most important phase, as granulation tissue becomes mature scar tissue over this time [23]. It is characterised by collagen remodelling and further contraction [47].

In the healing wound, granulation tissue is initially comprised of large amounts of type III collagen. During this phase, fibroblasts gradually change the type III collagen to collagen type I [23]. This leads to increased tensile strength of the scar. However, scar tissue will always remain weaker, as the final tensile strength is about 80% of that of surrounding healthy skin [32].

The collagen fibres additionally undergo some reorganisation. The collagen that is initially laid down is thinner than that in uninjured skin and is orientated parallel to the skin (instead of the basket weave pattern seen in uninjured skin) [25]. Over time, the collagen fibers are reabsorbed and deposited thicker, rearranged and cross-linked, such that they align along mechanical tension lines [47]. The latter contrasts with the random alignment of collagen fibres in healthy ECM [32].

As granulation tissue matures into scar tissue, the cell densities decrease. Many of the cells undergo *apoptosis* (i.e. programmed cell death). This leaves a relatively acellular and avascular, flat and thin scar of gradually increasing strength [45]. The extracellular matrix has now successfully been restored, albeit with slightly different properties than the pre-injured ECM.

2.4. Burn Injuries to the Skin

A burn is an injury to the skin usually caused by heat. Since burns are a type of wound, they heal in a similar way as described in Section 2.3. However, one distinction between general wound healing is that in the case of a burn, hemostasis is often bypassed. The reason for this is that intense heat can cause coagulation and destruction of blood vessels, leading to reduced or almost immediate cessation of bleeding [27]. As a result, the hemostasis phase is not as prominent or may not occur at all in many burn injuries. That is why, when a burn occurs, the body's response is to initiate the inflammatory phase of wound healing directly [17].

Severe burns can lead to a significant decrease in mobility in the affected area over the long term, primarily due to the development of *contractures* and *hypertrophic scarring*. As detailed in Section 2.3, contraction is a natural response of the body, facilitating healing. However, when excessive, contractions may become pathological [14]. If long-term reduced mobility occurs, it is commonly named a contracture [17]. The degree of contracture severity is influenced by factors such as the size of the wound, its location on the body and the extent of the skin tightening [83]. Contractures can inflict significant pain and discomfort on the patient and may lead to lifelong disabilities that can profoundly impact their future.

Another complication often arising in severe burn injuries is hypertrophic scarring. This pathological condition is characterised by a stiff, raised and uneven-textured scar, that does not extend far from the edges of the original wound [32]. Hypertrophic scarring is due to excessive healing, where an excess of ECM is produced and deposited [14]. In wound healing that leads to pathological scars, the inflammatory response is often greater and continues for an unusually long period of time. During the proliferative phase, fibroblasts and myofibroblasts continue to proliferate and synthesise ECM components much longer than usual, possibly due to this prolonged inflammation [32]. This leads to hypertrophic scarring, which can inflict pain, discomfort, and itch on the patient. Hypertrophic scars can also restrict movement if they are located close to a joint [22].

3

Morphoelastic Model for Burn Injuries

This chapter presents the mathematical model for skin evolution after burn trauma. The mathematical framework is outlined in Section 3.1, starting with the general system of equations and then providing a more detailed description of the relevant biological and mechanical components present in the model. Section 3.2 briefly discusses the numerical solver used for solving the system of equations and lastly, Section 3.3 defines an important measure for contraction: the *relative surface area* (of the) *wound* (RSAW).

3.1. Mathematical Framework

The two-dimensional morphoelastic model for post-burn wound contraction was developed by Koppenol [46], who used the theory of *morphoelasticity* developed by Hall [32] to incorporate the formation of long term deformations (contraction) into the dermal layer of the skin.

The model considers four biological constituents and three mechanical components as the primary variables. The biological constituents are the fibroblasts (N), the myofibroblasts (M), a generic signalling molecule (c), and collagen (ρ). The mechanical components are the dermal layer displacement $(\mathbf{u})$, the dermal layer displacement velocity $(\mathbf{v})$, and the effective strain $(\boldsymbol{\varepsilon})$. The following system of partial differential equations is used as a basis for the model:

$$\frac{\mathrm{D}z_i}{\mathrm{D}t} + z_i(\nabla \cdot \mathbf{v}) = -\nabla \cdot \mathbf{J}_i + R_i, \tag{3.1}$$

$$\rho_t \left(\frac{\mathrm{D}\mathbf{v}}{\mathrm{D}t} + \mathbf{v}(\nabla \cdot \mathbf{v}) \right) = \nabla \cdot \boldsymbol{\sigma} + \mathbf{f}, \tag{3.2}$$

$$\frac{\mathrm{D}\boldsymbol{\varepsilon}}{\mathrm{D}t} + \boldsymbol{\varepsilon}\,\mathsf{skw}(\nabla \mathbf{v}) - \mathsf{skw}(\nabla \mathbf{v})\,\boldsymbol{\varepsilon} + (\mathrm{tr}\,(\boldsymbol{\varepsilon}) - 1)\,\mathrm{sym}(\nabla \mathbf{v}) = -\mathbf{G}. \tag{3.3}$$

Firstly, we explain some notation. The operator $\frac{\mathrm{D}}{\mathrm{D}t}$ stands for the *material derivative* [3]:

$$\frac{\mathrm{D}}{\mathrm{D}t} \equiv \frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla.$$

If the material derivative is applied to a second-order tensor, it is applied to each of the scalar elements of this tensor separately.

Secondly, any second-order tensor $\mathbf{L}$ may be written as the sum of a symmetric and skew-symmetric part in the following manner:

$$\mathbf{L} = \frac{1}{2}\left(\mathbf{L} + \mathbf{L}^T\right) + \frac{1}{2}\left(\mathbf{L} - \mathbf{L}^T\right) = \mathrm{sym}(\mathbf{L}) + \mathrm{skw}(\mathbf{L}).$$

Here, $\mathrm{sym}(\mathbf{L})$ is a second-order tensor with the property $\mathrm{sym}(\mathbf{L}) = \mathrm{sym}(\mathbf{L})^T$ and $\mathrm{skw}(\mathbf{L})$ is a second-order tensor with the property $\mathrm{skw}(\mathbf{L}) = -\mathrm{skw}(\mathbf{L})^T$ [91].

Equation (3.1) is the conservation equation for the cell density or concentration for each of the four biological constituents. Here z_i represents the concentration, $\mathbf{J}_i$ is the flux per unit area, and R_i is a reaction term representing the kinetics of constituent i, for $i \in \{N, M, c, \rho\}$. A more precise expression for $\mathbf{J}_i$ and R_i for each of the constituents will be presented in Sections 3.1.1 to 3.1.4.

Equation (3.2) is the conservation equation for linear momentum. Here ρ_t represents the total mass density of the dermal tissue, $\boldsymbol{\sigma}$ is the stress tensor, and $\mathbf{f}$ is the total body force working on the dermal layer. Note that $\mathbf{v} = \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t}$. The constitutive relation for $\boldsymbol{\sigma}$, as well as a more precise expression for $\mathbf{f}$ will be formulated in Section 3.1.5. We note that Equation (3.2) actually gives rise to multiple equations, one for each component of the velocity vector $\mathbf{v}$.

Lastly, Equation (3.3) is the evolution equation that describes how the infinitesimal effective strain ($\boldsymbol{\varepsilon}$) changes over time. It is this equation that captures the morphoelasticity of the dermal layer, taking into account permanent deformation (in this case contraction) and residual stresses. It was formulated by Hall [32] and is based on his extensive theory on the *zero stress state* and morphoelasticity. For a precise description, we refer the reader to Hall's dissertation [32]. The second-order tensor $\mathbf{G}$ in Equation (3.3) is a growth tensor that describes the rate of active change of the effective strain, which will be formulated in Section 3.1.5. We note that Equation (3.3) actually gives rise to multiple equations, one for each component of the strain tensor $\boldsymbol{\varepsilon}$.

3.1.1. Fibroblast Population

In order to simplify notation, we replace z_i by i. Hence, z_N becomes N, the cell density of the fibroblasts in the dermis. Let us first describe the appropriate flux-term $\mathbf{J}_N$, that incorporates both random movement of fibroblasts through the dermal layer and the directed movement of fibroblasts up the gradient of signalling molecule c, if present. The former is modelled by a cell density dependent Fickian diffusion [75], and the latter process is modelled using a simple model for chemotaxis [36]. Taken together this gives

$$\mathbf{J}_N = -D_F F \nabla N + \chi_F N \nabla c,$$

where $F = N + M$. Here D_F is the (myo)fibroblast diffusion parameter and χ_F is the chemotactic parameter.

Equation (3.1) additionally contains a reaction term R_N describing the kinetics of the fibroblasts. Three factors are taken into consideration: proliferation, differentiation into myofibroblasts, and apoptosis. The first is modelled using an adjusted logistic growth model. The presence of a signalling molecule c is assumed to enhance both proliferation and cell differentiation. We obtain the following expression:

$$R_N = r_F \left(1 + \frac{r_F^{\max} c}{a_c^I + c}\right)(1 - \kappa_F F) N^{1+q} - k_F c N - \delta_N N.$$

Here, the parameter r_F is the cell division rate, $r_F^{\max}$ is the maximum factor with which the cell division rate can be enhanced due to the presence of the signalling molecule, a_c^I is the concentration of the signalling molecule that causes the half-maximum enhancement of the cell division rate. $\kappa_F F$ represents the reduction in the cell division rate due to crowding, q is a fixed constant, k_F is the signalling molecule-dependent cell differentiation rate of fibroblasts into myofibroblasts, and δ_N is the apoptosis rate of fibroblasts.

3.1.2. Myofibroblast Population

For the myofibroblasts, the flux-term in Equation (3.1) in very similar to the one for the fibroblasts. In the same way, it accounts for random movement of myofibroblasts through the dermal layer and the directed movement of myofibroblasts up the gradient of signalling molecule c. We obtain:

$$\mathbf{J}_M = -D_F F \nabla M + \chi_F M \nabla c.$$

The reaction term describing the kinetics of myofibroblasts is very similar as well. The same adjusted logistic growth model is used, the only difference being the assumption that myofibroblasts solely divide when the generic signalling molecule is present. This gives us the following:

$$R_M = r_F \left(\frac{(1 + r_F^{\max}) c}{a_c^I + c}\right)(1 - \kappa_F F) M^{1+q} - k_F c M - \delta_M M,$$

where δ_M is the apoptosis rate of myofibroblasts.

3.1.3. Generic Signalling Molecules

We assume that the signalling molecules diffuse through the dermis according to linear Fickian diffusion. This gives the following flux-term:

$$\mathbf{J}_c = -D_c \nabla c,$$

where D_c is the diffusion coefficient of the generic signalling molecule.

Furthermore, we assume that both fibroblasts and myofibroblasts release and consume the signalling molecules. Additionally, signalling molecules are removed from the dermis through proteolytic breakdown (breakdown of proteins into smaller components). The reaction term becomes the following:

$$R_c = \frac{k_c(N + \eta^I M)c}{a_c^{II} + c} - \delta_c \, g(N, M, c, \rho)c.$$

Here, k_c is the maximum net secretion rate of the signalling molecule, η^I is the ratio of myofibroblasts to fibroblasts in the maximum net secretion rate of the signalling molecules and the collagen molecules, a_c^{II} is the concentration of the signalling molecule that causes the half-maximum net secretion rate of the signalling molecule, and δ_c is the proteolytic breakdown rate of the signalling molecule.

The function $g(N, M, c, \rho)$ represents the concentration of a generic *metalloproteinase* (MMP). This enzyme is assumed to remove the signalling molecules through a proteolytic breakdown. In this study, we take the following relationship:

$$g(N, M, c, \rho) = \frac{(N + \eta^{II} M)\rho}{1 + a_c^{III} c}.$$

The parameter η^{II} is the ratio of myofibroblasts to fibroblasts in the secretion rate of the MMPs and the $1/(1 + a_c^{III} c)$ term represents the inhibition of the secretion of the MMPs due to the presence of the signalling molecule.

3.1.4. Collagen Molecules

For collagen, we assume that there is no active transport in the dermis, as secreted collagen molecules are attached to the ECM instantly. This means that the flux-term in Equation (3.1) is zero:

$$\mathbf{J}_\rho = \mathbf{0}.$$

For the reaction term, three variables are incorporated: collagen molecules are produced by both fibroblasts and myofibroblasts, the secretion rate is enhanced in the presence of the signalling molecule, and there is a proteolytic collagen breakdown analogous to the removal of the signalling molecules. This collectively results into

$$R_\rho = k_\rho \left(1 + \frac{k_\rho^{\max} c}{a_c^{IV} + c}\right)(N + \eta^I M) - \delta_\rho \, g(N, M, c, \rho)\rho.$$

Here, k_ρ is the collagen molecule secretion rate, $k_\rho^{\max}$ is the maximum factor with which the secretion rate can be enhanced due to the presence of the signalling molecule, a_c^{IV} is the concentration of the signalling molecule that causes the half-maximum enhancement of the secretion rate, and δ_ρ is the degradation rate of the collagen molecules.

3.1.5. Mechanical Components

In Equation (3.2), a visco-elastic constitutive relation is used for the stress-strain relation in the dermal layer. The visco-elastic relation for the dermal stress is:

$$\boldsymbol{\sigma} = \mu_1 \mathrm{sym}(\nabla \mathbf{v}) + \mu_2 [\mathrm{tr}(\mathrm{sym}(\nabla \mathbf{v}))\mathbf{I}] + \frac{E\sqrt{\rho}}{1+\nu}\left(\boldsymbol{\varepsilon} + \mathrm{tr}(\boldsymbol{\varepsilon})\frac{\nu}{1-2\nu}\mathbf{I}\right).$$

Here μ_1 and μ_2 are the shear and bulk viscosity, respectively, and ν is the Poisson's ratio. $E\sqrt{\rho}$ represents the Young's modulus (stiffness), which we assume to be dependent on the concentration of the collagen molecules.

Additionally, the total body force $\mathbf{f}$ in Equation (3.2) needs a more precise description. We assume that the myofibroblasts generate an isotropic stress, due to their pulling on the ECM, which is proportional to the product of the cell density of the myofibroblasts and a simple function of the concentration of the collagen molecules:

$$\begin{aligned}\mathbf{f} &= \nabla \cdot \boldsymbol{\psi},\\ \boldsymbol{\psi} &= \xi M \left(\frac{\rho}{R^2+\rho^2}\right)\mathbf{I}.\end{aligned}$$

$\boldsymbol{\psi}$ is a second-order tensor representing the total generated stress by the myofibroblast population, the parameter ξ is the generated stress per unit cell density and the inverse of the unit collagen concentration, and R is a fixed constant.

Lastly, we consider the growth contribution tensor $\mathbf{G}$ in Equation (3.3). We assume that the rate of active change of the effective strain is proportional to four factors: the product of the amount of effective strain, the local concentration of the MMPs, the local concentration of the signalling molecule, and the inverse of the local concentration of the collagen molecules. Taken collectively, this results into the following symmetric tensor:

$$\mathbf{G} = \zeta\left(\frac{g(N,M,c,\rho)c}{\rho}\right)\boldsymbol{\varepsilon} = \zeta\left(\frac{(N+\eta^{II}M)c}{1+a_c^{III}c}\right)\boldsymbol{\varepsilon},$$

where the parameter ζ is the rate of morphoelastic change.

3.1.6. Boundary Conditions

The domain of computation is defined by $\Omega_{\mathbf{x}}$ and the boundary of the computational domain by $\overline{\Omega}_{\mathbf{x}}$. No boundary conditions are specified for ρ and $\boldsymbol{\varepsilon}$ to avoid over-determination, as the equations for ρ and $\boldsymbol{\varepsilon}$ are ordinary differential equations for time t.

Let $\overline{\Omega}_{\mathbf{x}} = \Gamma^o_{\mathbf{x}} \bigcup \Gamma^h_{\mathbf{x}} \bigcup \Gamma^v_{\mathbf{x}}$, where Γ^o represents the outer non-symmetrical boundaries, Γ^h represents the horizontal symmetrical boundary where $y = 0$, and Γ^v represents the vertical symmetrical boundary where $x = 0$. For the chemicals, the following boundary conditions hold for all time t and all

$$\begin{aligned}&\mathbf{x} \in \Gamma^o_{\mathbf{x}}: \quad N(\mathbf{x},t) = \overline{N}, \quad M(\mathbf{x},t) = \overline{M}, \quad \text{and} \quad c(\mathbf{x},t) = \overline{c},\\ &\mathbf{x} \in \Gamma^p_{\mathbf{x}}: \quad \mathbf{J}_{N/M/c} \cdot \mathbf{n} = 0,\end{aligned}$$

where $p \in \{h, v\}$ and $\mathbf{n}$ is the outward pointing normal vector. We use similar conditions for the mechanics, for all time t and all

$$\begin{aligned}&\mathbf{x} \in \Gamma^o_{\mathbf{x}}: \quad \mathbf{v}(\mathbf{x},t) = 0,\\ &\mathbf{x} \in \Gamma^p_{\mathbf{x}}: \quad \mathbf{v} \cdot \mathbf{n} = 0 \quad \text{and} \quad (\sigma \cdot \mathbf{n}) \cdot \tau = 0,\end{aligned}$$

where τ is the tangential vector.

3.1.7. Initial Conditions

The initial conditions describe the cell densities and concentrations at the start of the proliferative phase of wound healing. Signalling molecules are present in the wound due to their secretion during the inflammatory phase of wound healing. Initially, fibroblasts and collagen are assumed to be present, whereas myofibroblasts are assumed to be absent.

The initial wounded area is denoted by $\Omega^w(0) \subset \Omega_{\mathbf{x},0}$. The unwounded area is then $\Omega_{\mathbf{x},0} \setminus \Omega^w(0)$. Let $d(\mathbf{x})$ be the shortest distance from a point $\mathbf{x} \in \Omega^w(0)$ to the wound boundary. Let $\Omega^w_s = \{\mathbf{x} \in \Omega^w(0) : d(\mathbf{x}) \geq s\}$. Then, for $z \in \{N, c, \rho\}$ we have the following initial densities or concentrations:

$$z(\mathbf{x},0) = \begin{cases}\tilde{z}, & \mathbf{x} \in \Omega^w_s,\\ \overline{z}, & \mathbf{x} \in \Omega_{\mathbf{x},0} \setminus \Omega^w(0),\end{cases}$$

where $\tilde{z}, \overline{z} \in \mathbb{R}^+$, the latter indicating the equilibrium value. Furthermore,

$$M(\mathbf{x},t) = \overline{M} = 0, \quad \mathbf{x} \in \Omega_{\mathbf{x},0}.$$

To create a smooth transition between the wound and unwounded area, the wound boundary steepness is modelled using half a sine wave for N, c, ρ. The initial conditions for the mechanical part of the model are all equal to zero for $\mathbf{x} \in \Omega_{\mathbf{x},0}$.

3.2. Numerical Solver

The mathematical model is solved using the finite element method with linear basis functions, with the implementation provided in Matlab. For the time integration, the backward Euler method is employed and to handle non-linearity, inner Picard iterations are used. Additional details on the implementation are omitted, as they are not crucial to this work. They can be found in [21]. Numerical computations are performed on a reduced, symmetrical domain to minimise computational workload, with the solution naturally inheriting this symmetrical property. A precise description of the domain and initial wound will be given in the subsequent chapter.

3.3. Relative Surface Area Wound

During healing, due to myofibroblasts pulling on the surrounding collagen fibers, the wound contracts towards its center and retracts after these cells disappear. The relative surface area (of the) wound (RSAW) is an important measure, as it gives valuable information about contraction. It is defined as follows:

$$\text{RSAW}(t) = \frac{area(\Omega^w(t))}{area(\Omega^w(0))}.$$

In the numerical model, the RSAW is determined as a post-processing step. The area of the wound at time t can be determined from the displacement field, by locating the positions of the wound boundary grid points at that time, and computing the area of the convex hull. The RSAW gives insight into key aspects of wound healing: its minimum indicates the point of maximal contraction. Once this minimum is reached, the wound starts to retract and grows in size. Eventually, after remodelling, the RSAW reaches an asymptotic value that represents the final fixed percentage of scar contraction. Figure 3.1 visualises a typical RSAW distribution over time.

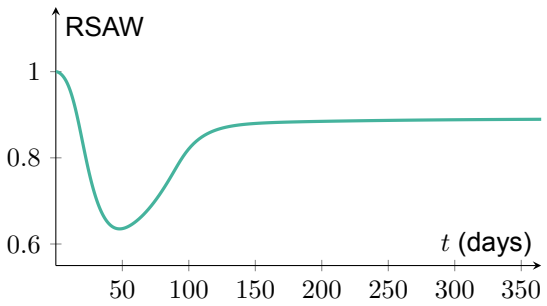


Figure 3.1: Example of a typical RSAW distribution over time. The minimum RSAW corresponds to maximal contraction, while the asymptotic value corresponds to final contraction.

4

Machine Learning Fundamentals

The function of this chapter is twofold: firstly, it serves to give an introduction into machine learning and specifically artificial neural networks. Section 4.1 gives a comprehensive overview of neural networks, covering all relevant concepts and terms. This section is based on the following sources: [7], [29], and [35]. Secondly, this chapter introduces a particular neural network family in Section 4.2 that is central to this study: neural operators. The focus will be on deep operator networks (DeepONets), which we will apply ourselves in the subsequent chapters to predict skin evolution.

4.1. A Brief Overview of Neural Networks

This section gives an introduction into neural networks. Section 4.1.1 gives a general architecture description. Section 4.1.2 covers activation functions used to introduce non-linearity in the network. Training of the network is explained in Section 4.1.3, followed by the explanation of forward propagation and backpropagation in Section 4.1.4 and Section 4.1.5, respectively. Different optimisation algorithms are considered in Section 4.1.6, and testing and validation of a network is discussed in Section 4.1.7. Lastly, Section 4.1.8 defines performance metrics used to evaluate trained models.

4.1.1. General Architecture and Description

Neural networks are computational models designed to mimic the brain's learning process. They consist of interconnected nodes, or *neurons*, each performing simple computations [29]. The neurons are organised in layers, including an *input layer*, one or more *hidden layers*, and an *output layer*. A hidden layer is called *dense* or *fully-connected* if each neuron in the layer is connected to all neurons in the subsequent layer. The number of layers in the network is called the *depth*. That is why the term *deep learning* or *deep neural network* refers to networks with multiple hidden layers. The *width* of the network is defined as the number of neurons in the hidden layers. Figure 4.1 provides a visualisation of a fully-connected, deep neural network, also called a *multilayer perceptron* (MLP).

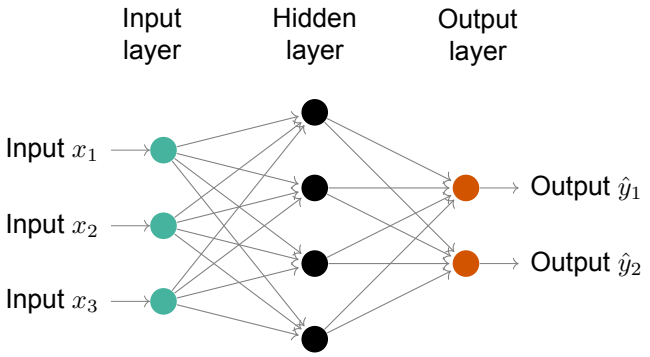


Figure 4.1: A visualisation of a fully-connected, deep neural network.

A neural network is used to approximate a mapping $\mathbf{f}$ that takes as input a vector $\mathbf{x} \in \mathbb{R}^n$ and outputs a vector $\mathbf{y} \in \mathbb{R}^m$, such that $\mathbf{y} = \mathbf{f}(\mathbf{x})$. It has been shown that neural networks are universal function approximators. This means that, with enough hidden layers, a neural network is able to approximate any (nonlinear) continuous function $\mathbf{f}$ arbitrarily well [29]. A neural network is essentially a parametrised mapping $\hat{\mathbf{f}}$ that takes as input $\mathbf{x} \in \mathbb{R}^n$ and outputs $\hat{\mathbf{y}} \in \mathbb{R}^m$. It tries to learn the values of certain parameters $\boldsymbol{\theta}$, such that $\hat{\mathbf{f}}(\mathbf{x};\boldsymbol{\theta}) \approx \mathbf{f}(\mathbf{x})$, i.e., $\hat{\mathbf{y}} \approx \mathbf{y}$.

In order to approximate both linear and nonlinear mappings, a neural network uses a combination of an affine transformation and a nonlinear *activation function*. Let us assume we have a fully-connected network with L hidden layers. Each neuron in the first hidden layer takes as input the entire vector $\mathbf{x}$ and applies the affine transformation $\mathbf{g}(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{g}(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T\mathbf{x} + b$. Here $\mathbf{w} \in \mathbb{R}^n$ is a vector of the *weights* of the neuron and $b \in \mathbb{R}$ is the *bias*. If we combine these weights and biases in a matrix $W_1 \in \mathbb{R}^{n_1 \times n}$ and $\mathbf{b}_1 \in \mathbb{R}^{n_1}$ respectively for all n_1 neurons in this hidden layer, the transformed vector $\hat{\mathbf{x}}$ can be computed as

$$\hat{\mathbf{x}} = W_1\mathbf{x} + \mathbf{b}_1.$$

The transformed vector is subsequently passed through a fixed nonlinear activation function α, to obtain the hidden values of the first layer:

$$\mathbf{h}_1 = \alpha(\hat{\mathbf{x}}) = \alpha(W_1\mathbf{x} + \mathbf{b}_1).$$

Note that the activation function is applied element-wise to the vector $\hat{\mathbf{x}}$, but itself is still a scalar function. There are many choices for an activation function. They will be discussed in greater detail in Section 4.1.2.

The output $\mathbf{h}_1 \in \mathbb{R}^{n_1}$ of the first hidden layer is taken as input of each node in the second hidden layer and the same process is repeated. This continues through all layers of the network. To summarise, a neural network with L hidden layers can be written as

$$\mathbf{h}_1 = \alpha(W_1\mathbf{x} + \mathbf{b}_1), \tag{4.1}$$
$$\mathbf{h}_{i+1} = \alpha(W_{i+1}\mathbf{h}_i + \mathbf{b}_{i+1}), \quad \text{for } i = 1, \dots, L-1 \tag{4.2}$$
$$\hat{\mathbf{y}} = \beta(W_{L+1}\mathbf{h}_L). \tag{4.3}$$

Here $\hat{\mathbf{y}} \in \mathbb{R}^m$ is the output of the neural network. Note that for the output layer there is often a different activation function used, denoted by β. One may also add a bias vector to the last layer. The vectors $\mathbf{h}_i \in \mathbb{R}^{n_i}$ for $i = 1, \dots L$ are the respective outputs of the hidden layers, which can be interpreted as the intermediate states of the network. The matrices $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and vectors $\mathbf{b}_i$, for $i = 1, \dots L+1$ contain the weights and biases, respectively. These are collectively called the *learnable parameters* of the network, which are learned through a process called *training*. Section 4.1.3 will cover the training of learnable parameters.

If we define

$$\begin{aligned}
\mathbf{F}_1 &:= \alpha(W_1\mathbf{x} + \mathbf{b}_1), \\
\mathbf{F}_i &:= \alpha(W_{i+1}\mathbf{h}_i + \mathbf{b}_{i+1}), \quad \text{for } i = 1, \dots, L-1 \\
\mathbf{F}_L &:= \beta(W_{L+1}\mathbf{h}_L),
\end{aligned}$$

we can formulate the neural network as the composition of parameter-dependent functions:

$$\hat{\mathbf{f}}(\mathbf{x};\boldsymbol{\theta}) = \mathbf{F}_L \circ \cdots \circ \mathbf{F}_1(\mathbf{x}). \tag{4.4}$$

Here $\boldsymbol{\theta}$ denotes all learnable parameters.

4.1.2. Activation Functions

Activation functions introduce non-linearity into neural networks, enabling them to approximate nonlinear functions and thereby capture intricate relationships within data [29]. Without activation function, the output of the neural network would be a linear combination of the inputs, which greatly restrict the usability. Common activation functions include the *sigmoid*, *tanh* and *Rectified Linear Unit* (ReLU) functions, which are visualised in Figure 4.2.

The ReLU function is defined by

$$\alpha(x) = \max(0, x).$$

The advantage of the ReLU function is that it is computationally efficient. A disadvantage is that it can cause nodes to "die", since the gradient for $x < 0$ is always zero. This implies that backpropagation is no longer possible and learning can terminate. More on backpropagation can be found in Section 4.1.5. There exist adaptations of the ReLU activation function, designed to prevent the problem described above. One of them is the *Leaky ReLU*, defined in the following manner:

$$\alpha(x) = \max(0.1x, x).$$

The sigmoid function is defined by

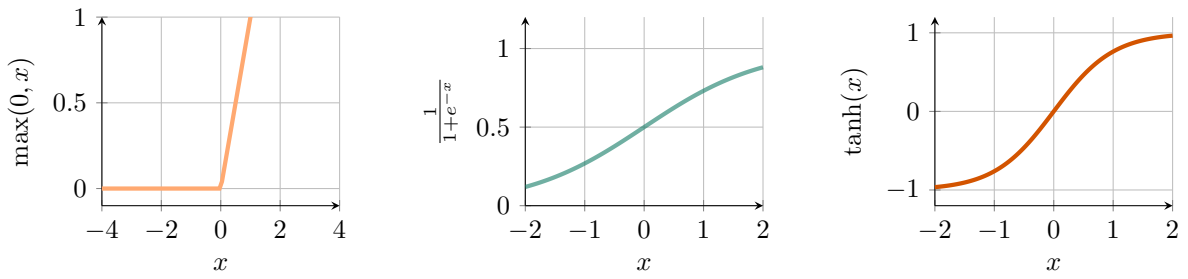
$$\alpha(x) = \frac{1}{1 + e^{-x}}.$$

Advantages of the sigmoid function are that it normalises the outputs of the hidden layers and it has a smooth gradient. A large drawback is that it suffers from vanishing gradients if the values of $|x|$ are large. This can slow down or even terminate learning. For this reason, the sigmoid function is not often used as activation function in hidden layers. However, it is useful in the output layer of classification models, where it returns a probability (a score between 0 and 1).

The hyperbolic tangent can be written as

$$\alpha(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

The $\tanh$ has the same properties as the sigmoid function, except that it is zero-centered, making it more suitable for inputs with strongly negative, neutral, and strongly positive values.



(a) ReLU **(b)** Sigmoid **(c)** $\tanh$

Figure 4.2: Three common activation functions.

Each activation function has its own strengths and weaknesses. When training a neural network, it is important to experiment with different activation functions to determine which one yields the best performance for the specific task.

4.1.3. Loss Functions and Training

Let I and O be the input and output sets, respectively:

$$I = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}, \qquad O = \{\mathbf{y}^1, \ldots, \mathbf{y}^N\}, \tag{4.5}$$

where $\mathbf{x}^i \in \mathbb{R}^n$ and $\mathbf{y}^i \in \mathbb{R}^m$, for $i = 1, \ldots, N$. Section 4.1.1 described the perspective of a neural network as a function $\hat{\mathbf{f}}$ that maps an input vector $\mathbf{x}^i \in I$ to an output vector $\hat{\mathbf{y}}^i \in \mathbb{R}^m$. As illustrated in Equation (4.4), this function is actually a composition of parameterised functions, allowing us to express it as $\hat{\mathbf{f}}(\mathbf{x}^i; \boldsymbol{\theta}) = \hat{\mathbf{y}}^i$. The aim of the neural network is to approximate the possibly non-linear, but continuous function $\mathbf{f}$ that has the same inputs $\mathbf{x}^i \in I$ but outputs the corresponding $\mathbf{y}^i \in O$. The neural network performs well if $\hat{\mathbf{y}}^i \approx \mathbf{y}^i$, for all $i = 1, \ldots, N$

Its objective is to find the values of the weights and biases, collectively denoted by $\boldsymbol{\theta}$, such that deviations of the network output from the desired output are penalised. Training a neural network means solving the following general optimisation problem:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}), \mathbf{y}^i\right). \tag{4.6}$$

Here $\mathcal{L}$ is a general *loss function*, which defines a distance metric between the network output and the desired output $\mathbf{y}$. There are various choices for $\mathcal{L}$, with one popular example being the *mean squared error* (MSE), defined as

$$\mathcal{L}^{MSE}\left(\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}), \mathbf{y}^i\right) = \frac{1}{N}\left|\left|\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}) - \mathbf{y}^i\right|\right|_2^2.$$

Other well-known loss functions include the *root mean squared error* (RMSE) and *mean absolute error* (MAE):

$$\mathcal{L}^{RMSE}(\left(\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}), \mathbf{y}^i\right)) = \frac{1}{\sqrt{N}}\left|\left|\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}) - \mathbf{y}^i\right|\right|_2,$$
$$\mathcal{L}^{MAE}(\left(\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}), \mathbf{y}^i\right)) = \frac{1}{N}\left|\left|\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}) - \mathbf{y}^i\right|\right|_1.$$

The set I defined in Equation (4.5) is often called the *training set*, containing the *training data*. One element of the training set is called a *training sample*. During training, the training data is passed through the network multiple times to adapt or train the learnable parameters $\boldsymbol{\theta}$. Training consists of three phases: *forward propagation*, *backpropagation*, and *optimisation*. The former two will be treated in Section 4.1.4 and Section 4.1.5, respectively. The latter refers to solving Equation (4.6), which usually involves solving a complex, highly non-convex optimisation problem. This is realised using a gradient-based optimisation algorithm. Section 4.1.6 will delve into this subject in greater detail.

4.1.4. Initialisation and Forward Propagation

The first phase in training is the computation of the *predictions* $\hat{\mathbf{y}}^i = \hat{\mathbf{f}}(\mathbf{x}^i, \boldsymbol{\theta})$ for each sample in the training set. This is called forward propagation. For a given training sample, the prediction can be calculated using the scheme detailed in Equations (4.1) to (4.3): the input values are fed to the first layer of the network, multiplied by its weights and added to its bias, and then passed through a nonlinear activation function before being passed to the next layer. This process is repeated until the output layer is reached, giving the prediction based on the current set of weights and biases in the network.

These weights and biases need to be initialised in order to compute the first predictions. This is done once at the beginning of training, and subsequently, the parameters are shared across all training samples during forward propagation. Initialisation can be done in multiple ways and often depends on the network architecture considered. One common technique is *Kaiming* or *He initialisation* [34], where the weights are drawn from a normal distribution:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_j}\right),$$

where n_j is the number of inputs of the j-th layer. Kaiming initialisation is the default method in PyTorch and is particularly effective in neural networks with ReLU activations.

Another common technique is *Xavier initialisation* [28]. Here the weights are drawn from the uniform distribution:

$$W \sim \mathcal{U}\left(\frac{-\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right),$$

where n_j is the number of inputs of the j-th layer and n_{j+1} is the number of outputs of the $(j+1)$-th layer. The biases are often initialised with zeroes.

After forward propagation, the loss function $\mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}), \mathbf{y}^i\right)$ between the predictions and actual values can be evaluated for each training sample. By adding all terms we can now formulate the minimisation problem that needs to be solved, as given in Equation (4.6).

4.1.5. The Computational Graph and Backpropagation

To solve Equation (4.6), the gradient of the sum of loss functions with respect to the learnable parameters $\boldsymbol{\theta}$ needs to be determined. Due to linearity, the gradients of the separate terms in the objective function can be calculated individually and then summed. This means that the network needs to determine

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i;\boldsymbol{\theta}), \mathbf{y}^i\right),$$

for $i = 1, \dots N$. Note that the learnable parameters $\boldsymbol{\theta}$ consist of the weight matrices W_j and bias vectors $\mathbf{b}_j$, for $j = 1, \dots, L+1$ (for each hidden layer in the network and the output layer).

During training of a neural network, the numerical evaluation of a gradient expression needs to be determined many times, which can be computationally expensive. The *backpropagation algorithm* provides an easy and computationally cheap solution, making efficient use of the chain rule for differentiation. It works by recursively applying the chain rule to compute the gradient of the loss with respect to each layer's output. These gradients are then propagated backward through the network to determine the gradients with respect to the parameters in each layer.

For illustrative purposes, let $\mathcal{L}(\hat{\mathbf{y}}) \in \mathbb{R}$, $\hat{\mathbf{y}} \in \mathbb{R}^m$, $\boldsymbol{\theta} \in \mathbb{R}^n$, and $\hat{\mathbf{y}} = g(\boldsymbol{\theta})$. The chain rule for differentiation is then given by

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{j=1}^{m} \frac{\partial \mathcal{L}}{\partial y_j} \frac{\partial y_j}{\partial \theta_i}.$$

Hence, the gradient of $\mathcal{L}$ with respect to $\boldsymbol{\theta}$ can be written as

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \left(\frac{\partial \hat{\mathbf{y}}}{\partial \boldsymbol{\theta}}\right)^T \nabla_{\hat{\mathbf{y}}} \mathcal{L},$$

where $\frac{\partial \hat{\mathbf{y}}}{\partial \boldsymbol{\theta}}$ denotes the Jacobian matrix of function g. Now assume we have $\mathcal{L}(\hat{\mathbf{y}}) \in \mathbb{R}$, $\hat{\mathbf{y}} \in \mathbb{R}^m$, $\mathbf{h} \in \mathbb{R}^k$, $\boldsymbol{\theta} \in \mathbb{R}^n$, $\hat{\mathbf{y}} = g(\mathbf{h})$, and $\mathbf{h} = l(\boldsymbol{\theta})$. The gradient of $\mathcal{L}$ with respect to $\boldsymbol{\theta}$ now becomes

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \left(\frac{\partial \mathbf{h}}{\partial \boldsymbol{\theta}}\right)^T \left(\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}}\right)^T \nabla_{\hat{\mathbf{y}}} \mathcal{L}.$$

Here $\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}}$ and $\frac{\partial \mathbf{h}}{\partial \boldsymbol{\theta}}$ denote the Jacobian matrices of function g and l, respectively. This principle of obtaining the gradient of the loss function with respect to the network's learnable parameters as a product of its gradient and Jacobian matrices at each network layer, is at the backbone of the backpropagation algorithm.

One way to conceptualise this better is by considering the *computational graph* of the neural network. Here each variable and operation (addition, multiplication, the application of an activation function) is denoted by a node in the graph. The graph keeps track of the order in which these operations are performed. Let us consider a simple example of a network with one input $x \in \mathbb{R}$, one hidden layer with one node, and one output $\hat{y} \in \mathbb{R}$. We consider only one sample of the training set. The output of the first hidden layer can be determined by

$$h = \alpha(w_1 x + b). \tag{4.7}$$

Assuming that the network's output is calculated using no activation function and no bias, we have

$$\hat{y} = w_2 h. \tag{4.8}$$

This gives rise to a loss function $\mathcal{L}(\hat{y}, y)$, where $y \in \mathbb{R}$ is the desired output. The computational graph of this simple neural network is depicted in Figure 4.3.

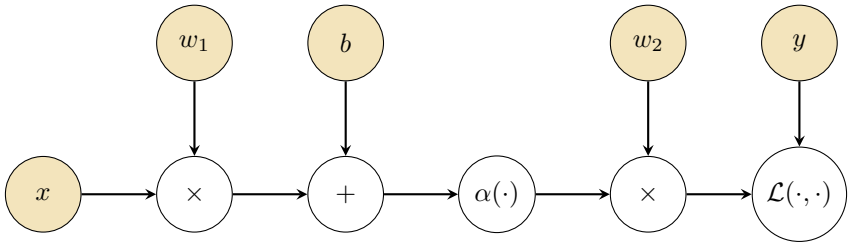


Figure 4.3: The computational graph of a simple neural network with one input $x \in \mathbb{R}$, one hidden layer with one neuron and one output $\hat{y} \in \mathbb{R}$. The white nodes in the computational graph denote operations, whereas the yellow nodes denote variables.

Sweeping through the computational graph from left to right gives the sequential order of operations in which we can break down Equation (4.7), Equation (4.8) and the calculation of the loss function $\mathcal{L}(\hat{y}, y)$. For each intermediate step, we can determine the partial derivatives of the output with respect to the inputs:

$$\begin{aligned}
f = w_1 x \quad &\implies \frac{\partial f}{\partial x} = w_1, \qquad \frac{\partial f}{\partial w_1} = x, \\
g = f + b \quad &\implies \frac{\partial g}{\partial f} = 1, \qquad \frac{\partial g}{\partial b} = 1, \\
h = \alpha(g) \quad &\implies \frac{\partial h}{\partial g} = \alpha'(g), \\
i = w_2 h \quad &\implies \frac{\partial i}{\partial h} = w_2, \qquad \frac{\partial i}{\partial w_2} = h, \\
j = \mathcal{L}(i, y) \quad &\implies \frac{\partial \mathcal{L}}{\partial i}, \qquad \frac{\partial \mathcal{L}}{\partial y}.
\end{aligned}$$

Subsequently, the derivative of the loss $\mathcal{L}$ with respect to the learnable parameters (in this case w_1, w_2, and b) is computed by traversing the computational graph in a reverse direction. This process involves multiplying the corresponding partial derivatives, effectively applying the chain rule multiple times:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial \mathcal{L}}{\partial i} \frac{\partial i}{\partial h} \frac{\partial h}{\partial g} \frac{\partial g}{\partial f} \frac{\partial f}{\partial w_1}, \\
\frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial \mathcal{L}}{\partial i} \frac{\partial i}{\partial w_2}, \\
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial i} \frac{\partial i}{\partial h} \frac{\partial h}{\partial g} \frac{\partial g}{\partial b}.
\end{aligned}$$

4.1.6. Optimisation Algorithms

After completing both forward propagation and backpropagation, the gradients of the objective function with respect to each learnable parameter are determined. These gradients can then be used in a gradient-based algorithm to find the updates of the learnable parameters.

Let us first introduce the standard *gradient descent algorithm* for updating the weights and biases of the network, also known as *batch gradient descent*. It is in line with our previous discussion of solving Equation (4.6) by determining $\nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i; \boldsymbol{\theta}), \mathbf{y}^i\right)$ for $i = 1, \ldots, N$, i.e, for all samples in the training set. The core idea is to utilise the fact that the gradient indicates the direction of steepest ascent of the objective function. This implies that the negative of the gradients points towards the direction of the function's minimum. Therefore, the gradient descent algorithm calculates the updates of the learnable parameters as follows:

$$\boldsymbol{\theta}_{updated} = \boldsymbol{\theta} - \lambda * \nabla_{\boldsymbol{\theta}} \left[\sum_{i=1}^{N} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i; \boldsymbol{\theta}), \mathbf{y}^i\right) \right]. \tag{4.9}$$

Here λ is the *learning rate*. This is a very important *hyperparameter* of the network that should be initialised by the researcher. Selecting a learning rate that is too large may cause the algorithm to

overshoot the minimum of the objective function, whereas choosing it too small can result in a very long computation time or prevent the algorithm from reaching a minimum at all.

After the learnable parameters are updated, the learning process recommences: forward propagation is applied to the entire training set, this time using the updated weights and biases. The new loss function that needs to be minimised is calculated and backpropagation is used do determine the gradient with respect to the updated parameters. Subsequently, the gradient descent algorithm updates the parameters once again, using the scheme in Equation (4.9). This process is repeated until a predefined stopping criterion is met. Once the network has finished training, the weights and biases of the last iteration are frozen.

In standard gradient descent, the entire training dataset is used to compute the gradient of the objective function with respect to the model parameters in each training step. This approach can become very computationally expensive, especially with large training sets. To address this, *mini-batch gradient descent* is commonly used. In this method, the training set is randomly divided into smaller, disjoint subsets of a fixed size. Such a set is called a *mini-batch* and the number of samples in a batch is a hyperparameter often denoted by the *batch size*. The gradient of the objective function (which is separable, as it consists of summed terms) is then approximated using only the terms corresponding to the samples in a single mini-batch. Each mini-batch contributes to one update of the model's parameters. We say that one *epoch* is completed after the model has iterated through all the mini-batches in the training dataset exactly once.

For illustrative purposes, let us assume we subdivide the training set into K disjoint sets $S_1, \dots, S_K$ with batch size k. We then approximate

$$\nabla_{\boldsymbol{\theta}} \left[\sum_{i=1}^{N} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i; \boldsymbol{\theta}), \mathbf{y}^i \right) \right] \approx \nabla_{\boldsymbol{\theta}} \left[\sum_{i \in S_j} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i; \boldsymbol{\theta}), \mathbf{y}^i \right) \right] = \sum_{i \in S_j} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i; \boldsymbol{\theta}), \mathbf{y}^i \right).$$

The mini-batch gradient descent algorithm calculates the updates of the learnable parameters as follows:

$$\boldsymbol{\theta}_{updated} = \boldsymbol{\theta} - \lambda * \nabla_{\boldsymbol{\theta}} \left[\sum_{i \in S_j} \mathcal{L}\left(\hat{\mathbf{f}}(\mathbf{x}^i; \boldsymbol{\theta}), \mathbf{y}^i \right) \right]. \tag{4.10}$$

The network iterates over the mini-batches, each time performing forward propagation and calculating the loss. It then performs backpropagation to compute the gradients of the loss with respect to the learnable parameters. Subsequently, the learnable parameters are updated using the scheme in Equation (4.10). This process is repeated for the desired number of epochs. For each new epoch, the data is randomly shuffled to ensure that the model does not learn patterns that might be specific to the order of the data in the training set, and new groups of mini-batches are created.

There exist many optimisation algorithms that are based on mini-batch stochastic gradient descent, but utilise a adaptive learning rate λ. Popular examples include *AdaGRAD* [16], *RMSprop* [37], *Adam* [43] and *NAdam* [15]. While mini-batch gradient descent uses a fixed learning rate for all parameters, these optimisers adjust the learning rates based on the historical behavior of each parameter during training. This approach is particularly useful in dealing with different scales and is shown to be more robust [84].

4.1.7. Validation and Testing

After the training phase, the neural network is ready to be evaluated on unseen data to assess its generalisation capabilities. This phase is crucial for determining how well the model performs on data it has never seen during training. The evaluation process involves freezing the weights and biases obtained from the final iteration of the training phase and subjecting the neural network to a *test dataset*.

To ensure a fair evaluation, it is important to distinguish between a *validation set* and a *test set*. Typically, the original dataset is divided into three subsets: the training set, the validation set, and the test set. The training set is used to update the model's parameters. The validation set is used during training to track performance on unseen data, tune hyperparameters, and make choices about the model's architecture. The test set, on the other hand, is reserved for evaluating the final model and provides an unbiased measure of its performance. The division ratio can be chosen by the researcher and often

depends on the amount of data available. A common split is 70% for training, 15% for validation, and 15% for testing, but these percentages can be adjusted based on specific needs.

Optimal performance often requires fine-tuning of hyperparameters, which are configuration settings that are not learned during training but can significantly influence the model's performance. Examples are the learning rate, the number of hidden layers, and the batch size. Techniques such as *grid search* or *random search* can be used to systematically explore the hyperparameter space and identify the combination that maximises the model's performance on the validation set [35].

4.1.8. Performance Metrics

There exist several metrics to gauge the performance of a neural network on the test set. Based on these values, the performance of different network architectures or settings can be compared. The performance measures we will consider in this study include the R^2 *statistic*, the *average relative root mean squared error* (aRRMSE), and *the average relative error* (aRelErr).

The R^2 score, also known as the coefficient of determination, is widely used to evaluate the performance of a regression model. It gives a measure of how well the predicted values approximate the actual values. Specifically, it indicates the proportion of the variance in the dependent variable (output) that is predicted from the independent variable (input). It is defined as follows:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n} \left(y^i - \hat{y}^i\right)^2}{\sum_{i=1}^{n} \left(y^i - \bar{y}\right)^2}, \tag{4.11}$$

where y^i denotes the true value, $\hat{y}^i$ denotes the predicted value, and $\bar{y}$ is the average true value, for $i = 1, \ldots, n$ samples in the test set. Note that a uniform average of the R^2 scores is taken when the network has multiple outputs. $R^2 = 1$ indicates a perfect fit, whereas $R^2 = 0$ implies that the model does not improve the prediction of simply returning the expected value of the observed values. A negative score can occur when the model performs worse than always predicting the mean.

A frequently used performance metric for multi-target regression is the aRelErr. It is an intuitive measure that gives an overall indication of how well the model performs across multiple targets in terms of the relative error. It is defined in the following way:

$$aRelErr = \frac{1}{d} \sum_{t=1}^{d} \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_t^i - \hat{y}_t^i}{y_t^i} \right|,$$

where d is the number of targets (network outputs). Note that the aRelErr is not a reliable measure for targets that have values close to zero, as the denominator then becomes very small, leading to a disproportionately large relative error, even if the predictions are accurate. To remedy this, we propose an adjusted definition of the aRelErr:

$$aRelErr = \frac{1}{d} \sum_{t=1}^{d} \frac{1}{n_t} \sum_{i=1}^{n} \mathbb{I}(y_t^i \neq 0) \cdot \left| \frac{y_t^i - \hat{y}_t^i}{y_t^i} \right|, \tag{4.12}$$

where

- y_t^i is the true value for the i-th sample and t-th target, rounded to one decimal place,
- $\hat{y}_t^i$ is the predicted value for the i-th sample and t-th target, rounded to one decimal place,
- $\mathbb{I}(y_t^i \neq 0)$ is the indicator function, which is 1 if $y_t^i \neq 0$ and 0 otherwise,
- n_t is the number of samples where $y_t^i \neq 0$ for the t-th target,
- d is the number of targets,
- n is the total number of samples.

The rounding ensures that the aRelErr does not blow up, as very small values are rounded to zero. The indicator function ensures that the relative error is only calculated for non-zero true values, avoiding division by zero.

Lastly, the aRRMSE is a different performance measure used for multi-target regression problems. It is a relative measure used in cases when the relative error proofs to be unsuitable and is defined as follows:

$$aRRMSE = \frac{1}{d}\sum_{t=1}^{d} RRMSE = \frac{1}{d}\sum_{t=1}^{d}\sqrt{\frac{\sum_{i=1}^{n}\left(y_t^i - \hat{y}_t^i\right)^2}{\sum_{i=1}^{n}\left(y_t^i - \bar{y}_t\right)^2}}, \tag{4.13}$$

where d is the number of targets. Despotovic et al. [11] have formulated interpretable ranges for the RRMSE:

- Model accuracy is considered excellent when $RRMSE < 0.1$,
- good when $0.1 < RRMSE < 0.2$,
- fair when $0.2 < RRMSE < 0.3$, and
- poor when $RRMSE > 0.3$.

4.2. Operator Learning

In Section 4.1.1, we have introduced neural networks as function approximators. Another and perhaps more powerful result is that a neural network with a single layer can accurately approximate any nonlinear *operator* (i.e., a mapping between infinite-dimensional function spaces) [8]. This leads to a different application named operator learning. Neural operators have the property of being discretisation-invariant [48]:

1. The model can act on any discretisation of the input function,
2. The model can be evaluated at any point of the output domain,
3. As the discretisation is refined, the model converges to a continuum operator.

Neural operators are highly effective for creating surrogate models for numerical solvers for partial differential equations (PDEs). The former can take functions as inputs, such as

- the right-hand side function of a PDE,
- the spatial distribution of a material parameter,
- a function describing a boundary condition, or
- a level-set function representing the geometry.

The output of the neural operator is then a function as well, representing the solution to the initial boundary value problem (IBVP). This approach contrasts with the more classical *physics-informed neural networks* (PINNs) [76], which usually only approximate the solution to a specific IBVP. PINNs are a class of neural networks that embed the physical laws governing the system (often in the form of PDEs, boundary conditions, or initial conditions) directly into the loss function. This allows them to solve forward and inverse problems in an efficient way, but they are designed to learn the solution of a specific (parametrised) PDE, rather than learning an operator that can generalise across different scenarios.

In recent years, several neural operator architectures have been emerged, with FNOs [50] and DeepONets [54] being the most widely used. Other architectures include *convolutional neural operators* (CNOs) [77], which maintain structure-preserving continuous-discrete equivalence, allowing for the learning of operators without introducing discretisation-dependent aliasing errors. The proposed CNO architecture is an extension of the popular U-Net architecture [81].

The focus in this study will be on DeepONets, for learning operators accurately and efficiently from a relatively small dataset. The subsequent section will treat DeepONets in greater detail.

4.2.1. DeepONets

DeepONets were introduced by Lu et al. [54] and their main idea is based on the universal approximation properties of operators composed of shallow neural networks. Let us first introduce some notation. We consider an operator G that takes an input function f, and then $G(f)$ is the corresponding output

function. For any point y in the domain of $G(f)$, the output $G(f)(y) \in \mathbb{R}$. The function f is evaluated at finitely many fixed locations $\{x_1, \ldots, x_m\}$, which Lu et al. call *sensors*.

Theorem 1 (Universal Approximation for Operators [8])**.** *Suppose that α is a continuous nonpolynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact subsets, respectively, V is a compact subset of $C(K_1)$, and G is a nonlinear continuous operator $V \to C(K_2)$. Then, for any $\varepsilon > 0$, there exist positive integers n, p, m and constants $c_i^k, w_{ij}^k, \beta_i^k, b_k \in \mathbb{R}, W_k \in \mathbb{R}^d, x_j \in K_1$, for $i = 1, \ldots, n$, $k = 1, \ldots, p$, and $j = 1, \ldots, m$, such that*

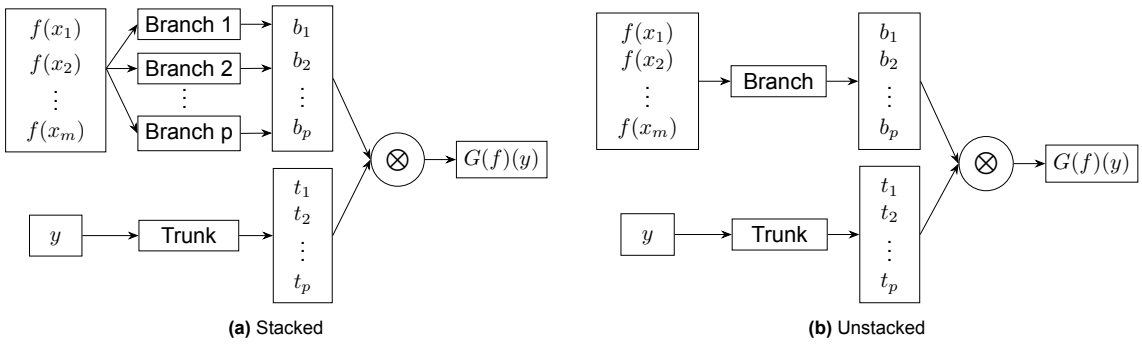
$$\left| G(f)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \alpha \left(\sum_{j=1}^{m} w_{ij}^k f(x_j) + \beta_i^k \right)}_{\text{branch}} \underbrace{\alpha(W_k y + b_k)}_{\text{trunk}} \right| < \varepsilon \tag{4.14}$$

for all $f \in V$ and $y \in K_2$.

This approximation theorem indicates the potential application of neural networks to learn nonlinear operators from data. The idea is to substitute what in Equation (4.14) is denoted by *branch* and *trunk* with a (deep) neural network, respectively. This leads to the general form of the DeepONet, as depicted in Figure 4.4a. Lu et al. refer to this architecture as the *stacked* DeepONet. The trunk network takes y as input and outputs a vector $(t_1, \ldots t_p) \in \mathbb{R}^p$. Additionally, there are p branch networks that are stacked in parallel, each taking as input $(f(x_1), \ldots f(x_m)) \in \mathbb{R}^m$ and having as output a scalar $b_k \in \mathbb{R}$, for $k = 1, \ldots p$. The output of the DeepONet is then the dot product

$$G(f)(y) \approx \sum_{k=1}^{p} b_k(x_1, \ldots, x_m) t_k(y). \tag{4.15}$$

Intuitively, we can think of Equation (4.15) as the basis representation of $G(f)(y)$, where the trunk net is responsible for learning the spatial basis functions $t_1, \ldots, t_p$, whereas the branch net provides the corresponding coefficients $b_1, \ldots, b_p$.



(a) Stacked **(b)** Unstacked

Figure 4.4: The two general DeepONet architectures as described in [54].

Lu et al. describe a second DeepONet architecture, where the p branches are merged into one single branch network. The single branch outputs a vector $(b_1, \ldots, b_p) \in \mathbb{R}^p$. This *unstacked* DeepONet is less computationally and memory expensive compared to the stacked version, and the authors even demonstrate that it often leads to a better performance and smaller generalisation errors. The unstacked DeepONet is schematically depicted in Figure 4.4b.

In the context of (a system of) PDEs, a DeepONet may be utilised to learn the solution operator of the system. Consider a system of PDEs where the solution is denoted by $u(x, t)$. Furthermore, there is a parameter $f(x)$ that the solution depends upon. There are many possibilities for f, examples including but are not limited to f representing a forcing term, a source term, an initial condition, some other variable parameter in the system, or the domain geometry. We can use a DeepONet to approximate the solution operator to the PDE:

$$G : f(x) \mapsto u(x, t).$$

That is: given any $f(x)$, the DeepONet predicts the solution $u(x,t)$ over the entire domain and for all time. The training set is generated by randomly sampling f from a chosen function space. Possible examples include *Gaussian random field* and *orthogonal (Chebyshev) polynomials* [54]. The only condition required is that the sensor locations $\{\hat{x}_1, \ldots, \hat{x}_m\}$ are the same for all input functions f^i. Then, for each f^i, the solution $u(x,t)$ is sampled at P random locations $\{(x_1,t_1)^i, \ldots, (x_P,t_P)^i\}$ in the domain. The training set for the DeepONet is then a triplet $[\mathbf{f}, \mathbf{y}, G(\mathbf{f})(\mathbf{y})]$, where

$$[\mathbf{f}, \mathbf{y}, G(\mathbf{f})(\mathbf{y})] = \left[\begin{bmatrix} \vdots \\ f^i(\hat{x}_1), \ldots, f^i(\hat{x}_m) \\ f^i(\hat{x}_1), \ldots, f^i(\hat{x}_m) \\ \vdots \\ f^i(\hat{x}_1), \ldots, f^i(\hat{x}_m) \\ \vdots \end{bmatrix}, \begin{bmatrix} \vdots \\ (x_1,t_1)^i \\ (x_2,t_2)^i \\ \vdots \\ (x_P,t_P)^i \\ \vdots \end{bmatrix}, \begin{bmatrix} \vdots \\ u^i(x_1,t_1) \\ u^i(x_2,t_2) \\ \vdots \\ u^i(x_P,t_P) \\ \vdots \end{bmatrix} \right].$$

If we assume an unstacked DeepONet, the branch network takes inputs from the first vector, the trunk network takes inputs from the second vector and the desired output to which we compare the network's output are elements from the third vector.

Note that there is no specific requirement for the architectures of the branch and trunk networks. Lu et al. make use of simple MLPs with two to four hidden layers. Many variants of DeepONets have since been introduced, where specific types of networks for branch and trunk are chosen. Examples are U-Net [13] and graph neural network (GNN) [89] architectures. Other approaches combine PINNs with DeepONets to produce *physics-informed DeepONets* [97]. The selection of trunk and branch architectures is typically based on the specific problem at hand, as well as the structure and dimensionality of the input functions.

5

Predictions on a Single Wound Shape

In this chapter, we address our first sub-question: can we train a DeepONet on one specific wound shape, to accurately predict the dermal displacement across the entire domain over time? Section 5.1 provides more specifics of the numerical model we use to compare our neural network against, and explains the rationale behind certain choices we make based on this model. The DeepONet architecture, as well as certain design choices, are discussed in Section 5.2. This is followed by Section 5.3, describing the approaches we use for generating datasets to train and test our model. Furthermore, Section 5.4 delves in the training procedure, explaining the chosen settings and hyperparameters. The trained DeepONet is put to the test in Section 5.5, considering its performance on the test set. Lastly, we formulate a conclusion in Section 5.6.

5.1. Numerical Simulations

In Chapter 3, we have seen that the numerical model solves for four biological constituents (N, M, c, ρ) and three mechanical components $(\mathbf{u}, \mathbf{v}, \boldsymbol{\varepsilon})$ over time. Furthermore, the RSAW and strain energy are computed as post-processing steps. Among these nine outputs, we consider the displacement $\mathbf{u}$ to be the most important one, as it provides direct information about contraction. From a clinician's point of view, predicting the wound's movement over time is particularly meaningful. This prediction offers insights into the timing and severity of contraction and retraction, which can be used to optimise treatment. Therefore, we choose to only predict the displacement $\mathbf{u}(t, x, y) = (u_1, u_2)$ of the dermal layer over time.

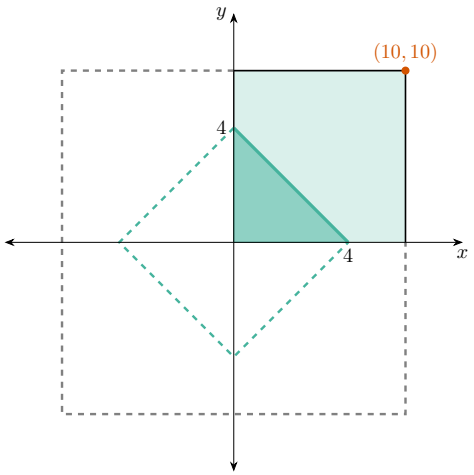


Figure 5.1: The rhombus-shaped initial wound geometry used in the numerical model. Only a quarter of the complete domain (coloured) is considered due to symmetry.

The domain of computation is the square $\Omega_{\mathbf{x}} = [-10, 10] \times [-10, 10]$ cm^2. The initial wound is a rotated square (rhombus) defined by the subset $|\frac{x}{4}|+|\frac{y}{4}| \leq 1$. Figure 5.1 provides a visualisation. The numerical computations are performed on a quarter of this domain, assuming symmetry in the x- and y-axis. The solution, particularly the displacement we are interested in, then also inherits this symmetrical property. The finite element model computes the displacement at each time step in 968 spatial points in the domain. These points are not uniformly distributed, since the triangular elements used increase in size as the distance from the wound boundary increases. As a result, the grid is densest around the wound boundary.

The inputs to the numerical scheme are 34 parameters from the morphoelastic model which can vary between simulations (i.e., patients). Please refer to Appendix A for a complete overview of the parameters and their values. Variations in the parameter values result in different solutions. We choose to not consider the effect of all parameters in predicting the displacement over time. This decision is driven by the ultimate focus of our research, which is to incorporate multiple wound shapes. Varying all parameters might introduce excessive variability in the solutions, making it more difficult for a neural network to learn patterns when multiple shapes are included. That is why, we restrict ourselves to varying five parameters. We select three (D_F, D_c, k_F) that appear to have a larger influence on the solution, and two (χ_F, a_c^I) with a smaller influence, based on a preliminary sensitivity study we performed. Egberts et al. [19] conducted a stability analysis of the mathematical model, formulating stability criteria in terms of specific parameters. In our selection, we ensured that no parameters are chosen that can easily disrupt these stability conditions. Table 5.1 gives the pre-described ranges from which we uniformly draw the five parameter values for each simulation. These ranges are based on a sensitivity study by Egberts et al. [21].

The numerical model simulates the solution over a chosen period in days, typically with $t_{end} = 365$ (one year). However, a one-year simulation may take up to 15 minutes, which is not favourable for creating large datasets. Therefore, we choose to let $t_{end} = 100$ days to reduce computation time. Empirically we found that after 100 days, a wound is typically still in the retraction phase and no asymptotic value is yet reached. This means that our decision to predict the displacement for $t \in [0, 100]$ will result in some information loss. However, we believe that this approach is still worthwhile, as the maximal contraction will be predicted and an indication of the asymptotic value can be derived.

Table 5.1: Ranges for the values of the five parameters that are varied in each finite element simulation.

Parameter	Range
D_F	$7.6167 \cdot 10^{-7} - 1.2 \cdot 10^{-6}$
χ_F	$(2 - 3) \cdot 10^{-3}$
D_c	$(2.22 - 3.2) \cdot 10^{-3}$
k_F	$8 \cdot 10^6 - 1.08 \cdot 10^7$
a_c^I	$(0.9 - 1.1) \cdot 10^{-8}$

5.2. DeepONet Architecture

We propose a neural operator, based on the unstacked DeepONet architecture, for predicting post-burn wound evolution. Specifically, the goal is to predict the displacement field $\mathbf{u}(t, x, y) = (u_1, u_2)$, given the values of the five parameters from the morphoelastic model. In Chapter 4, we have seen that the general architecture of a DeepONet is designed to output one single function. The first question we must answer is how to extend this to two functions u_1, u_2. Here we propose a few possible approaches:

1. We use two independent DeepONets, one for predicting the displacement in x-direction, and one for the y-direction.
2. We split the branch network in two and share the trunk network. This is depicted in Figure 5.2. Here we let the number of outputs of the branch be two times the number of outputs of the trunk. Then, we let the first group of the branch and the entire trunk generate u_1, and the second group and the trunk generate u_2.
3. We split the trunk network in two and share the branch network.

Although the first option is the simplest approach, we believe the two outputs we aim to predict are not independent. Since both represent displacements, in terms of writing them as a product of basis functions, it is reasonable to think they might share some characteristics. This could be either the basis functions themselves (which are predicted by the trunk network) or the coefficients (branch network). Therefore, we choose to consider either the second or third approach. Figure 5.3a shows the result of a small test we perform to compare the two. The exact training data sampling strategy and neural network settings will be discussed in the subsequent sections (see Table 5.2 for a quick overview). For now, it is important that we keep all hyperparameters constant and only vary the way we split the trunk/branch. Upon repeated execution, we find that splitting the branch results in a slightly lower validation loss after 150 epochs of training. Moreover, a lower validation loss is reached sooner. Based on these results, we decide in favour of splitting the branch network.

The exact architecture we adopt for our network is schematically depicted in Figure 5.2. For the branch and trunk networks, we take simple MLPs with three hidden layers, containing 50 neurons per layer. The branch takes as input the five chosen parameters from the morphoelastic model describing the wound contraction. Note that we take them to be constant over time and space. The output of the branch network is a vector $(b_1, \dots, b_{2p}) \in \mathbb{R}^{2p}$. The trunk takes as input a coordinate (t, x, y), in which we want to evaluate the displacement $\mathbf{u}(t, x, y) = (u_1, u_2)$. The output of the trunk network is a vector $(c_1, \dots, c_p) \in \mathbb{R}^p$. The output of the DeepONet is an inner product of the outputs of the trunk and branch networks. Specifically, if we let $T(\cdot)$ be the operation of the trunk and $B(\cdot)$ be the operation of the branch net, we may summarise it as follows:

$$
\begin{aligned}
c_1, \dots c_p &= T(t, x, y), \\
b_1, \dots b_{2p} &= B(D_F, \chi_F, D_C, k_F, a_C^I), \\
u_1(t, x, y) &= \sum_{i=1}^{p} b_i c_i, \\
u_2(t, x, y) &= \sum_{i=1}^{p} b_{i+p} c_i.
\end{aligned}
$$

therefore

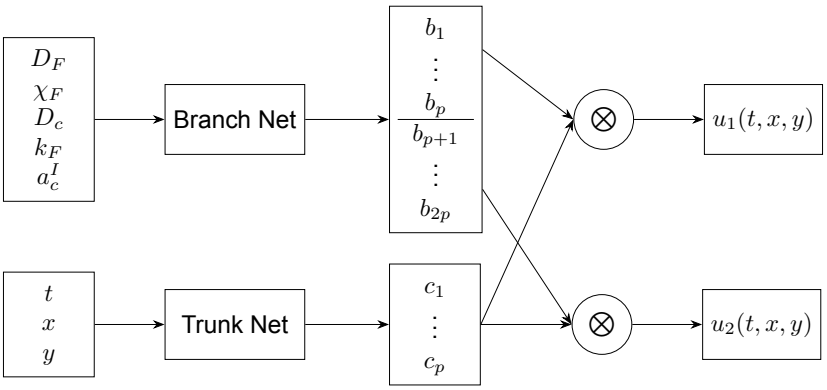
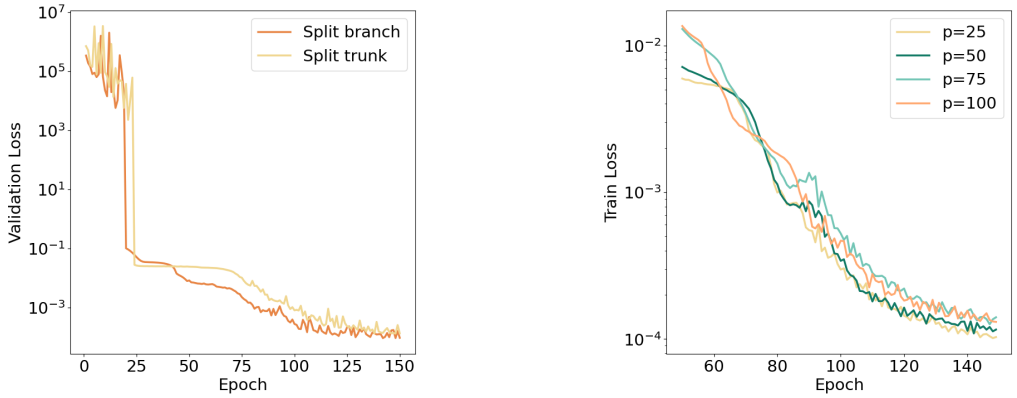


Figure 5.2: The initial DeepONet architecture for predicting dermal displacement.

The final design choice is determining p, the number of trunk outputs. We conduct a small test where we let $p \in \{25, 50, 75, 100\}$ and consider the respective train and validation losses thus obtained. Figure 5.3b plots the training losses against the number of epochs. The validation losses follow the exact same course, but are much noisier, so we chose not to display them here. We observe that increasing p does not result in a notable pattern. After 150 epochs, the final losses are all around 10^{-4}, the highest being 0.00014 (for $p = 75$) and the lowest being 0.00010 (for $p = 25$). Although this is a difference of 40%, we do not yet know what a loss of 10^{-4} means and if this is a significant difference. Furthermore, upon repeated execution, we find that the order of the curves is slightly different each time. This can be explained due to randomness in initialisation and data sampling. Since the test proves to be inconclusive, we opt for a middle ground and fix $p = 50$.



(a) Comparison of splitting the branch vs. splitting the trunk

(b) Training loss for different values of p

Figure 5.3: Two tests for determining the DeepONet architecture. In both cases, all other hyperparameters were kept fixed during training.

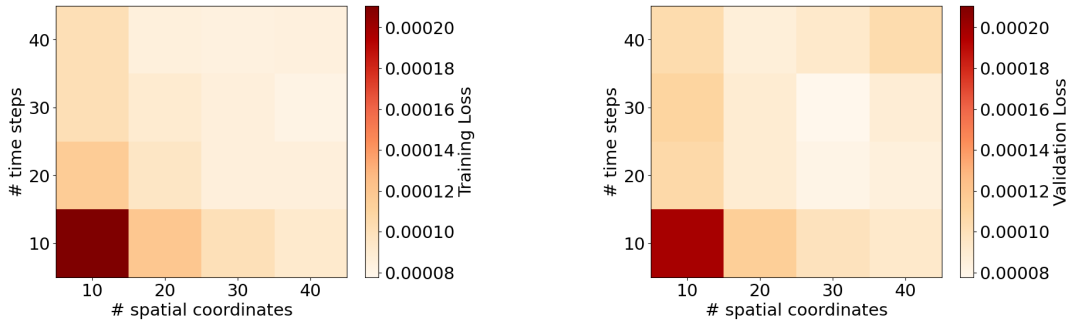
5.3. Datasets

For generating a training set, we initially run 150 finite element simulations. For each one, we fix $t_{end} = 100$ days. We opt for the following data sampling strategy: for each finite element simulation, m time steps are uniformly sampled. Note that the total number of time steps in $[0, 100]$ differs among simulations, as the numerical model uses an adaptive time stepping scheme based on how much retraction occurs. Then, for each drawn time step, n spatial coordinates from the domain (out of the total 968) are uniformly sampled. This gives us $m \cdot n$ coordinates (t, x, y) per simulation, which are fed to the trunk. For each numerical simulation, the five parameter values taken as input to the branch are uniformly chosen from pre-specified ranges, as prescribed in Table 5.1. The numerical model then determines the displacements (u_1, u_2) in the $m \cdot n$ coordinates, given the values of the five parameters. These are the truth values used to calculate the loss during training of the DeepONet. Following this procedure, each finite element simulation contributes the following triplet to the training data:

$$\left[\begin{bmatrix} D_F, \chi_F, D_c, k_F, a_c \\ D_F, \chi_F, D_c, k_F, a_c \\ \vdots \\ D_F, \chi_F, D_c, k_F, a_c \end{bmatrix}, \begin{bmatrix} t_1, x_1, y_1 \\ \vdots \\ t_1, x_n, y_n \\ \vdots \\ t_m, \hat{x}_1, \hat{y}_1 \\ \vdots \\ t_m, \hat{x}_n, \hat{y}_n \end{bmatrix}, \begin{bmatrix} u_1(t_1, x_1, y_1), u_2(t_1, x_1, y_1) \\ \vdots \\ u_1(t_1, x_n, y_n), u_2(t_1, x_n, y_n) \\ \vdots \\ u_1(t_m, \hat{x}_1, \hat{y}_1), u_2(t_m, \hat{x}_1, \hat{y}_1) \\ \vdots \\ u_1(t_m, \hat{x}_n, \hat{y}_n), u_2(t_m, \hat{x}_n, \hat{y}_n) \end{bmatrix} \right].$$

The first column is input to the branch, the second column is input to the trunk, and the third column represents the target values.

The question remains how to choose m and n. To get an indication, we perform a grid search where we consider $m, n \in \{10, 20, 30, 40\}$. We use the dataset containing 150 finite element simulations for training (20% is used for validation), where all possible sampling combinations are evaluated. For each combination, we train the network four times and take the average of the final losses (after 150 epochs). This approach accounts for the variability due to randomness inherent in the training procedure. Figure 5.4 displays the results.



(a) Training **(b)** Validation

Figure 5.4: Grid search investigating the best combination of sampling time steps and spatial coordinates. 150 finite element simulations are used for the dataset, using an 80-20 split for training and validation, respectively.

We observe that increasing the number of time steps (m) generally leads to a lower MSE loss. The same can be observed when increasing the number of spatial coordinates (n). This is particularly evident in the training loss, which is in line with our expectations: sampling more data (either more time steps, more coordinates, or both) results in a training set with more data points, making it easier for the model to learn patterns. However, this trend does not always hold when we consider the validation loss. For example, we note that the loss for $m, n = 40$ is not much lower than for $m = 10, n = 20$. This could be an indication of overfitting, where the model is exposed to too much information from each finite element run, making it difficult to generalise to new cases. On the other hand, we are still not sure whether the difference between a loss of 0.00008 and 0.0002 is meaningful in terms of actual predictions. We do plan to include more finite element simulations in the training data, which will naturally increase the total number of data point. Moreover, sampling twice as many data points per finite element simulation results in twice the training time. That is why, to err on the side of caution, we select $m = 10, n = 20$.

This means that the training set consisting of 150 finite element simulations now contains $150 \cdot 10 \cdot 20 = 30.000$ data points. Additionally, we generate five more data sets, each containing the same number of data points. The largest training set we can create from this is based on 750 independent finite element runs, resulting in a total of 150.000 data points. The final 150 samples we save for a test set to evaluate our trained model on.

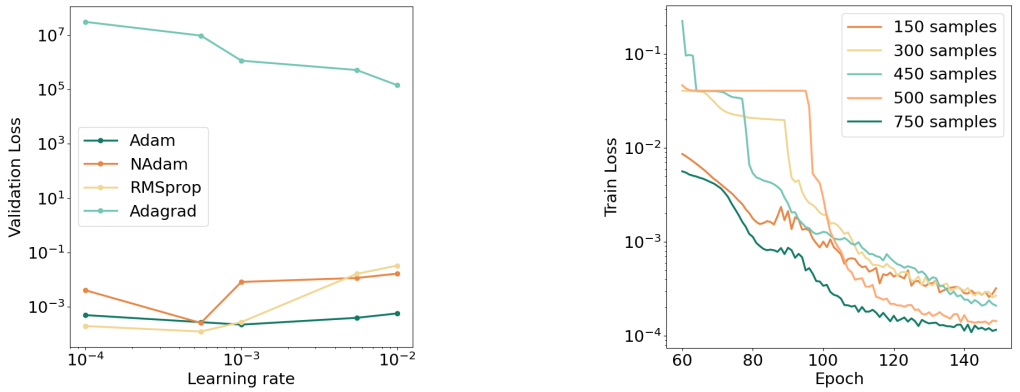
5.4. Training

In the previous sections, there was already mention of training the DeepONet and comparing the losses in different small tests. This section serves to give an overview of all the hyperparameters and settings we fixed upon, which are summarised in Table 5.2. There are many choices involved, typically supported by the results of hyperparameter tuning. We have already seen some examples thereof in Figure 5.3 and Figure 5.4. We do not consider the effect of varying all hyperparameters, as this is not the ultimate goal of our research. Therefore, we fix the MSE as loss function, use ReLU activation function, and adopt Pytorch's default (Kaiming) for initialising the weights and biases.

We do additionally investigate the optimiser and learning rate. Figure 5.5a plots different learning rates against the final validation loss after 150 epochs, for four optimisers. We use the smallest dataset based on 150 finite element runs for training. The training for each setting is repeated five times and an average is computed, to account for variability in the loss due to randomness. It is immediately clear that the Adagrad optimiser is not suitable for our case, since it results in very large losses across the whole range of learning rates. Ideally, we would like the learning rate to be as large as possible, while still maintaining a low loss. In this context, NAdam is not a competitor. If we compare RMSprop and Adam, we observe that the former achieves the lowest loss for a learning rate of $lr = 0.0005$. However, Adam exhibits the lowest trend across the entire range of learning rates. Consequently, we choose Adam with $lr = 0.001$.

Furthermore, we investigate the number of data points in the training set. Figure 5.5b illustrates the effects of systematically increasing (or decreasing) the number of samples. Note that with a sample in the training set we refer to the data of one finite element run, for which we uniformly draw $10 \cdot 20$ data

points. To ensure a fair comparison, we adapt the batch sizes such that the total number of iterations is constant over 150 epochs of training. We choose to display the training losses, since the validation losses follow the same patterns but with greater noise. Multiple runs reveal that increasing the number of samples consistently results in a lower achieved loss. After approximately 135 epochs, we have a decreasing order of the losses for an increasing number of samples. Notably, the largest dataset produces the lowest loss for all epochs. Since we find that the training time does not linearly scale with the number of samples (the largest dataset requires only 29% more time to train than the smallest), we decide that it is worthwhile to include all 750 samples.



(a) Learning rate vs. validation loss for multiple optimisers

(b) Effect of increasing the number of samples in the training set

Figure 5.5: Additional hyperparameter tuning for determining the optimiser, batch size, and number of data points.

We have now finalised all hyperparameters and settings. Figure 5.6 depicts the training and validation loss obtained from the largest dataset, which takes approximately 10 minutes to train. We observe that the losses stagnate around 10^{-4}, indicating that it is indeed sufficient to train for 150 epochs. It is now time to put the trained model to the test.

Table 5.2: DeepONet hyperparameters and settings for training on one initial wound shape.

Attribute	**Value / Setting**
Type NN	DeepONet
Type branch, trunk	MLP
No. inputs branch	5
No. inputs trunk	3
No. neurons in hidden layers	50/50/50
No. outputs branch	100
No. outputs trunk (p)	50
No. data points	30.000 to 150.000 (varies by dataset)
Train/validation split	80/20
Initialisation	Default (Kaiming)
Activation function	ReLU
Loss function	MSE
Optimiser	Adam
Learning rate	0.001
No. epochs	150
Batch size	20 to 100 (varies by dataset)
Total no. iterations	180.000

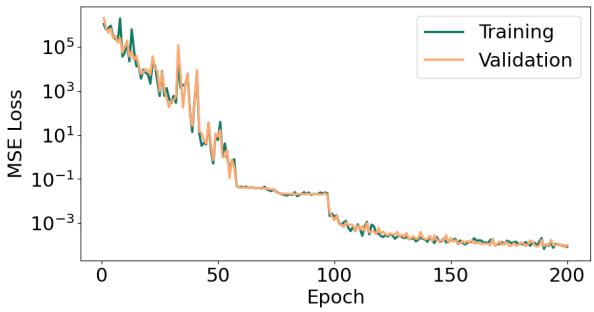
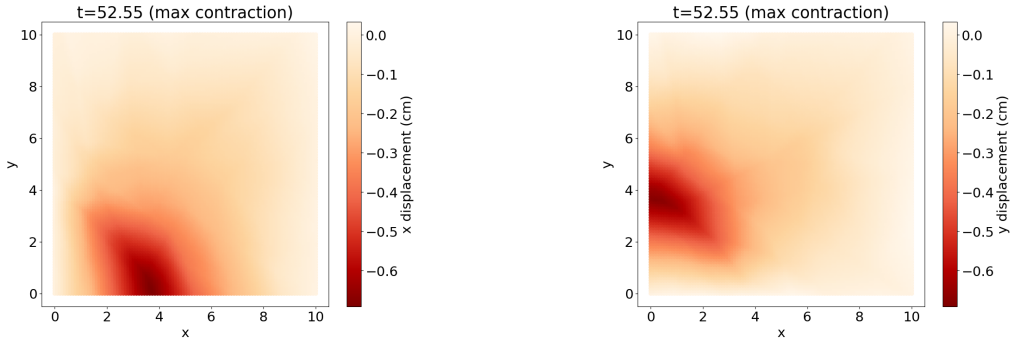


Figure 5.6: Training and validation losses obtained from the largest dataset. All other hyperparameters and settings are given in Table 5.2.

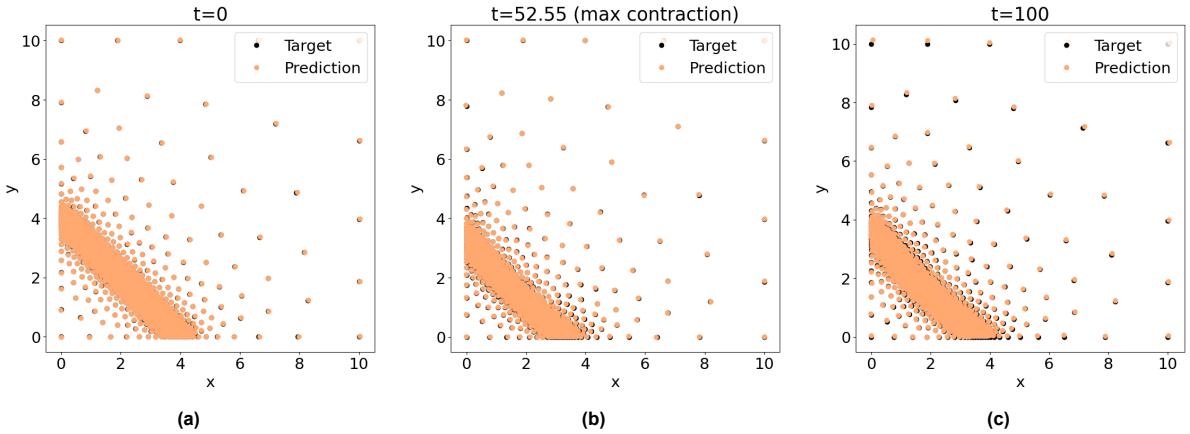
5.5. Performance on the Test Set

Our DeepONet predicts the entire displacement field at any given time t. We evaluate the trained model on each sample in the test set, which consists of data it has never seen during training. Figure 5.7 depicts the entire field prediction at the time of maximal contraction, for one sample in the test set. The displacement at a spatial coordinate is always taken as the difference between the current position and the initial position. Since the wound contracts, the displacements are negative. We observe that they are largest in magnitude at the wound boundary. As we move further away from the wound, displacements decrease, reaching zero at the top and right domain boundaries.



(a) x-direction **(b)** y-direction

Figure 5.7: Prediction of the whole displacement field in x and y-direction at maximal contraction for one sample from the test set. Corresponds to Figure 5.8b.

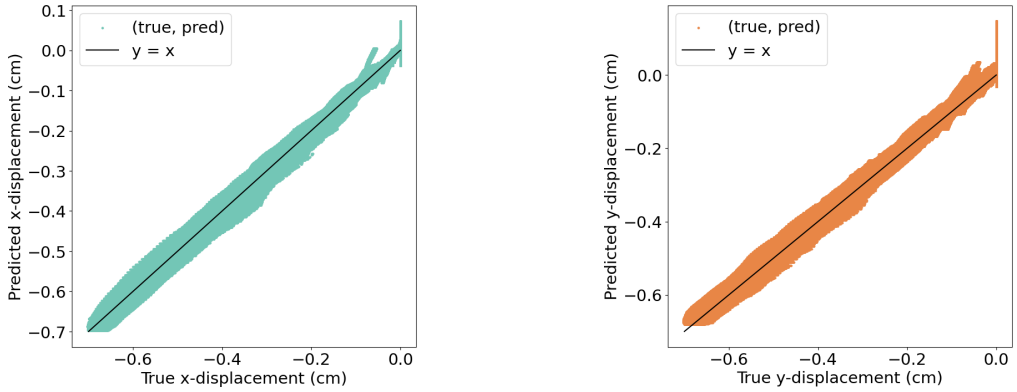


(a) **(b)** **(c)**

Figure 5.8: Prediction vs. target for one sample from the test set at $t = 0$, at the time of maximal contraction, and at $t = 100$.

To compare the predicted displacements with the numerical model, we consider the predictions at the 968 spatial points within the domain for the entire test set. Figure 5.8 shows one example of the obtained predictions versus the corresponding targets, at $t = 0$, at the time of maximal contraction, and at $t = 100$. Note that the actual coordinates are depicted here, rather than the displacements. We observe excellent predictive accuracy across all spatial points. It appears that as time increases, the predictions become slightly less accurate. This is particularly apparent at the boundaries of the domain at $t = 100$. There we even observe positive displacements, which should not occur. These observations are in accordance with the absolute error plots in Figure B.1 in Appendix B.

Further evidence of the model's predictive accuracy can be found in the scatter plots in Figure 7.3, where the true displacements are plotted against the predicted displacements. The line $y = x$ represents a perfect neural network. The plots contain all test set samples across all times steps: around 20.000.000 data points. We observe a strong concentration around the lines $y = x$, indicating that our model provides accurate predictions and has effectively learned the underlying patterns. However, there are notable outliers along the lines $x = 0$, indicating positive displacements where they should be zero. This occurs at the boundaries of the domain, since the boundary conditions are not explicitly enforced, and is in accordance with what we have observed in Figure 5.8. Furthermore, we notice that the model performs better at predicting smaller magnitude displacements in the range $[-0.3, 0]$ cm, based on the narrower bands around the lines $y = x$. These smaller magnitude displacements occur when t is small, supporting our observation that the predictions slightly degrade as time increases.



(a) x-direction **(b)** y-direction

Figure 5.9: True vs. predicted displacement in x- and y-direction on the test set.

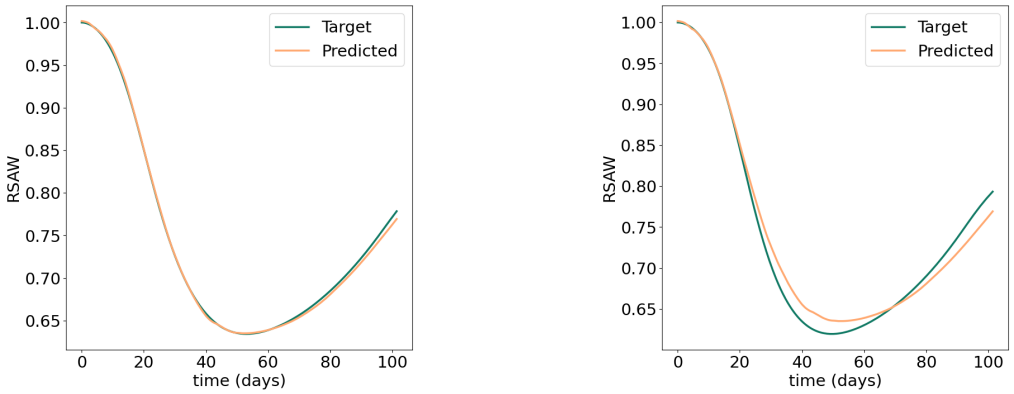
Table 5.3 presents the performance metrics of the DeepONet, including the R^2 score, the aRelErr, and the aRRMSE, which are defined in Equations (4.11) to (4.13), respectively. A value of $R^2 = 1$ indicates perfect predictions, whereas $R^2 = 0$ indicates that the model is not better than simply returning the expected values. For our DeepONet, we find $R^2 = 0.9975$, demonstrating that the model can very accurately predict dermal displacement. Furthermore, we find an aRRMSE of 0.0498, showing excellent performance according to Despotovic et al., as it is smaller than 0.1. [11]. Lastly, the aRelErr is found to be only 0.025, which further indicates that the model can excellently reproduce the finite element simulations.

Table 5.3: Performance of the DeepONet for predicting dermal displacement, trained on a rhombus-shaped wound.

Performance measure	Result on test set
R^2	0.99751
aRRMSE	0.04984
aRelErr	0.02500

Lastly, we evaluate the trained DeepONet specifcially on the wound boundary. The same test set is used, but only the 100 grid points on the wound boundary are considered at all time steps. This enables

us to calculate the RSAW over time. Figure 5.10 shows the best and worst prediction thus obtained. In the best-case scenario, the prediction is almost indistinguishable from the target, with only a slight underestimation of 1.2% of the RSAW at $t = 100$. In the worst-case scenario, target and prediction begin to diverge after 25 days. The model predicts a maximal contraction that is 2.5% less severe and occurs 3 days later than the target. Additionally, the final value at $t = 100$ is predicted to be 3% lower than the target. This indicates that the model predicts less retraction at this time than is actually present. However, since the model predicts displacements within the rough range of $[-0.7, 0]$ cm, a 3% error corresponds to deviations on the order of tenths of millimeters.



(a) Best prediction

(b) Worst prediction

Figure 5.10: Best and worst prediction (measured by MSE) on the test set in terms of the RSAW.

5.6. Conclusion

We began this chapter by posing the question of whether it is possible to train a DeepONet on a single wound shape, to accurately predict the dermal displacement across the entire domain over time. To investigate this, we trained a DeepONet with an architecture as illustrated in Figure 5.2, using the training settings given in Table 5.2. The training set contained data derived from a single rhombus-shaped initial wound of fixed size. The results in Section 5.5 demonstrate excellent predictive accuracy and performance. Consequently, we conclude that we can answer our research sub-question in the affirmative. Based on these positive outcomes, we will proceed with the current setup. A natural second step is to extend the training to multiple wound shapes with variable sizes.

6

Predictions on Multiple Wound Shapes

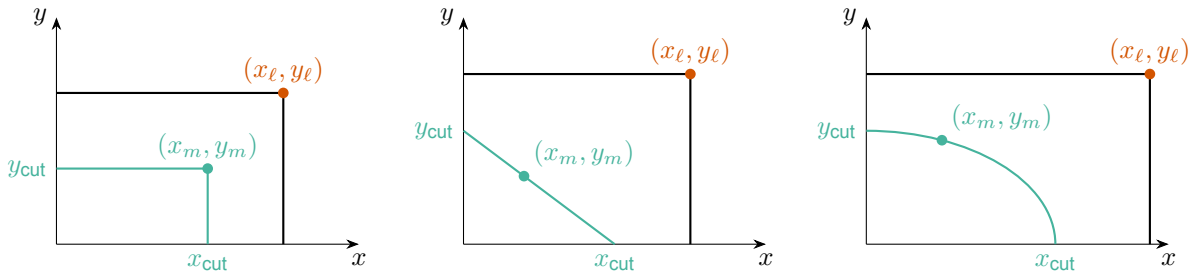
This chapter aims to answer our second sub-question: can we train a DeepONet on *multiple wound shapes*, to accurately predict the dermal displacement across the entire domain over time? Section 6.1 investigates the effects of extending and enriching our training set with multiple wound shapes, while keeping the current DeepONet architecture. Following this, Sections 6.2 to 6.4 aim to improve the obtained performance with extensions to the architecture. We consider, respectively, addition to the branch network, addition to the trunk network, and incorporation of a *sine augmentation* block. Finally, Section 6.5 summarises the findings.

6.1. Enrichment of the Dataset

This section describes the effects of enrichment to the dataset in the form of multiple wound shapes with variable sizes. This is introduced and explained in Section 6.1.1. Section 6.1.2 considers the performance of the DeepONet on a new test set.

6.1.1. Datasets and Training

The following step in answering our main research question is to train and test our DeepONet on multiple initial wound shapes with varying sizes, rather than just a single shape with fixed size. To this end, we use an updated version of the finite element code that can take three different initial geometries: a rectangle (including a square), a rhombus (including a rotated square), and an ellipse (including a circle). We again consider a quarter of the complete domain, assuming symmetry along the x- and y-axis, as shown in Figure 6.1.



(a) Rectangle **(b)** Rhombus **(c)** Ellipse

Figure 6.1: The three initial wound shapes used for training. Due to symmetry, a quarter of the complete domain is considered. Additionally, a visualisation of the points $(y_{cut}, x_m, y_m, x_{cut})$ and (x_ℓ, y_ℓ).

For generating the training set, we run 750 finite element simulations. In each run, we uniformly se-

lect one of the three initial wound shapes. For the wound size we uniformly draw $x_{cut}, y_{cut} \in (0, 5)$ centimeters. The size of the complete domain is then determined as

$$x_\ell = 2.5 \cdot x_{cut},$$
$$y_\ell = 2.5 \cdot y_{cut},$$

rounded to one decimal. For all simulations, we again fix $t_{end} = 100$ days. The training data sampling strategy is the same as before: for each finite element simulation, 10 time steps are uniformly sampled. Then, for each time step, 20 spatial coordinates from the domain are uniformly sampled. This gives us 200 coordinates (t, x, y) per simulation, in which we denote the displacements (u_1, u_2). Each triplet (t, x, y) is input to the trunk. In the same way as before, we uniformly pick the values of the five parameter that are input to the branch from pre-specified ranges given in Table 5.1. Following this procedure, we obtain a training set containing 150.000 data points. We use an 80-20 split for training and validation, respectively. For the test set, we execute an additional 150 finite element simulations.

The training setup is the same as before and is summarised in Table 5.2. This time, we train the network for 100 epochs, after which we find a train loss of 0.0082 and a validation loss of 0.0027. Training the model for more epochs does not further decrease the loss. Figure 6.2 shows a plot of the training and validation losses. We notice that after 50 epochs, both losses already stagnate around 10^{-2}. This is two orders of magnitude higher than the loss of 10^{-4} we observed previously, when training on only one wound shape (see Figure 5.6). We thus expect the performance of our current setup to not match our earlier results.

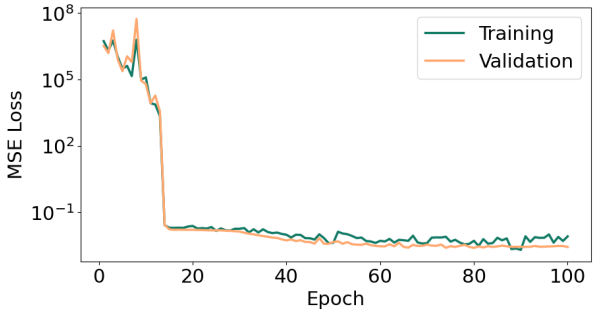


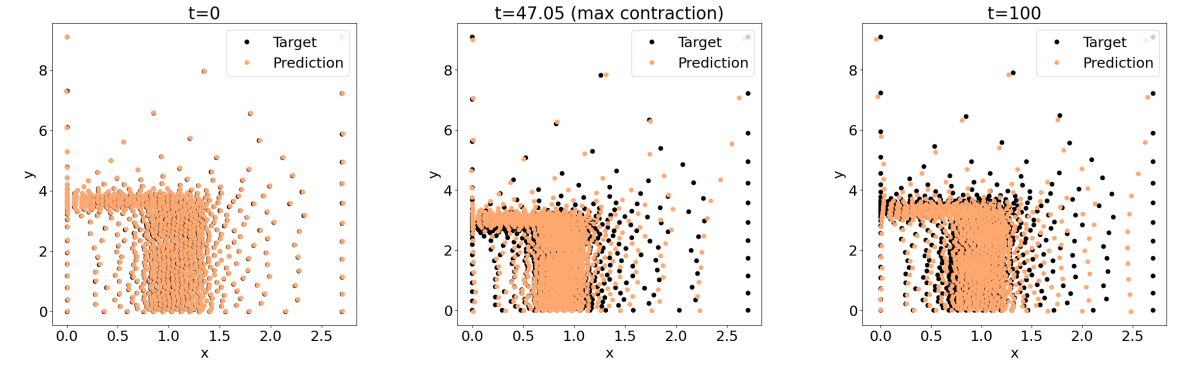
Figure 6.2: Training and validation losses.

6.1.2. Performance on the Test Set

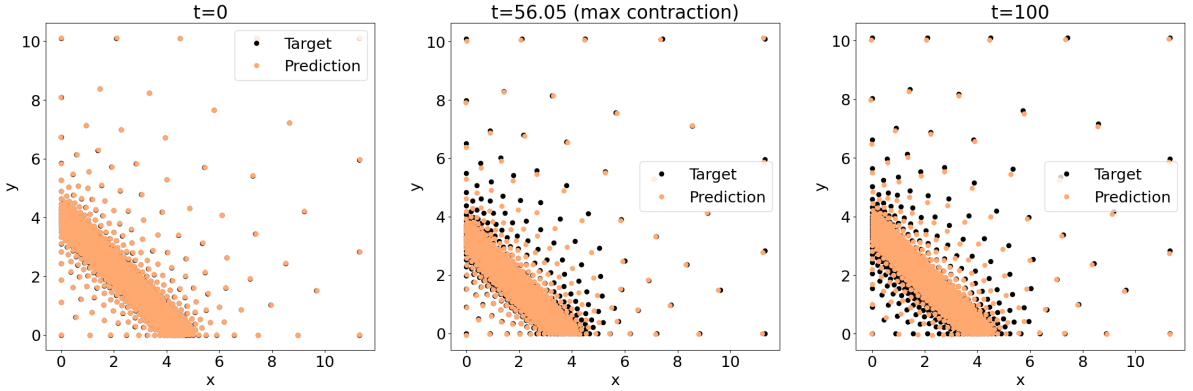
We evaluate our trained model on the test set that contains 150 samples not seen during training. Figure 6.3 shows three different examples of the obtained predictions versus the corresponding targets at $t = 0$, at their respective time of maximal contraction, and at $t = 100$. We observe that the model accurately predicts zero displacements at $t = 0$ for all shapes. As time increases, the prediction quality seems to decrease. At and directly surrounding the wound boundary, the predictions are best. Given that most of the spatial coordinates in the training set are located at the wound boundaries, it is expected that the network performs best in these regions. Further away from the wound, the prediction accuracy significantly decreases, where the model predicts displacement that are too large in magnitude. Additionally, the model seems to have great difficulties with predicting the displacements at the boundaries of the domain. These results are in accordance with the absolute error plots in Figure B.3.

Remarkably, the (boundary) predictions seem best for the rhombus-shaped wound, especially towards the top and right boundaries. For the rectangular-shaped wound, it appears to be the worst out of the three. The elliptic-shaped wound is somewhere in between. We have inspected the predictions of a large number of samples in the test set and have found a relation to the initial size of the wound (and hence the domain). It appears that the model has effectively learned that when a spatial point has 'large' x and y coordinates (meaning it is far away from the wound), the displacements are very small. However, the model has no way of knowing the size and shape of the wound beforehand (and thus implicitly the size of the domain). For example, if we consider the rectangular-shaped wound in

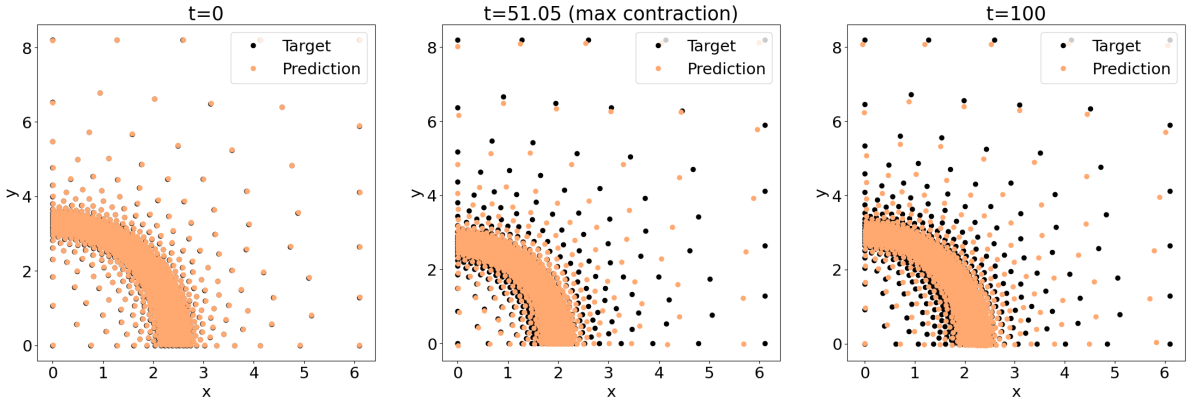
Figure 6.3, the right domain boundary is at $x = 2.7$. However, for other samples (for example the rhombus-shaped case below) this is at the wound boundary. Therefore, the network also predicts a large displacement in the former case. In other words: the network has overfitted to the largest wound shapes present in the training set and has no way of recognising smaller wounds.



(a) Rectangular-shaped wound



(b) Rhombus-shaped wound



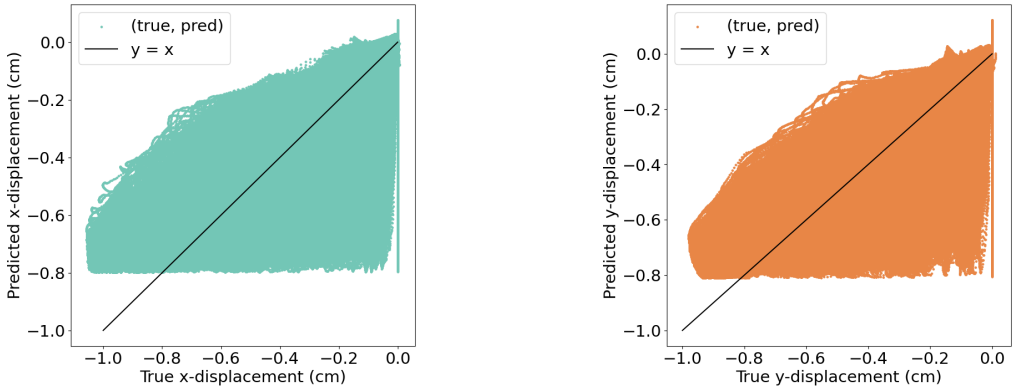
(c) Elliptic-shaped wound

Figure 6.3: Prediction vs. target for three samples from the test set at $t = 0$, at the time of maximal contraction, and at $t = 100$.

Table 6.1: Performance of the DeepONet for predicting dermal displacement, trained on three basic wound shapes.

Performance measure	Result on test set
R^2	0.89107
aRRMSE	0.32999
aRelErr	0.13582

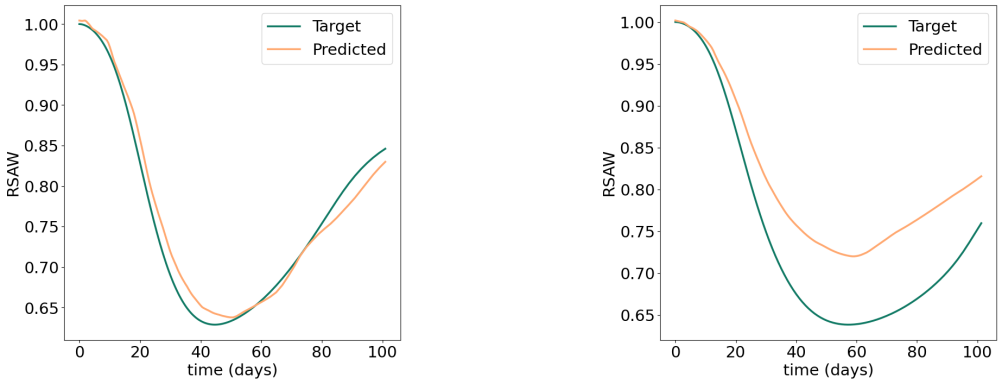
The lacking performance of our current model is apparent from Figure 6.4 and Table 6.1 as well. The former depicts scatter plots where the true displacements are plotted against the predicted displacements. It contains all test set samples, where we have taken all spatial coordinates at all time steps: a total of $18.316.361$ data points. We observe a very large spread of points around the line $y = x$, indicating that the model has not learned to accurately predict the displacements. The area underneath the line $y = x$ is somewhat larger than that above, which points to the model more often predicting displacements that are too large in magnitude. This is in accordance with what we have observed in Figure 6.3. Curiously, the largest displacements in magnitude in the rough range of $[-1, -0.8]$ cm are always greatly underestimated. These occur at at the wound boundary at the time of maximal contraction, for the largest wounds present in the test set. The performance metrics in Table 6.1 further demonstrate that the current DeepONet architecture is insufficient for accurately predicting dermal displacement when trained on the three initial wound shapes.



(a) x-direction **(b)** y-direction

Figure 6.4: True vs. predicted displacement in x- and y-direction on the test set.

Lastly, we evaluate our trained model on the wound boundary. We consider the same test set, but only the spatial coordinates on the wound boundary are taken. This enables us to determine the RSAW at each time step. Figure 6.5 shows the best and worst predictions. In the best case, we see an overestimation compared to the target in the first 51 days. Here, the model predicts a maximum of 3% less contraction. Additionally, it predicts a later maximal contraction (51 vs. 45 days) and retraction with less intensity. After day 51, we see an underestimation compared to the target. The final predicted value at $t = 100$ days is 1.9% lower than the target, indicating a larger final contraction. In the worst case, the model greatly overestimates the RSAW for all times. The minima of the two curves are obtained roughly on the same day, however, the DeepONet predicts 12.8% less maximal contraction and 7.4% less final contraction. It is evident that the current multi-shape model's performance is inadequate, indicating a need for improvement.



(a) Best prediction

(b) Worst prediction

Figure 6.5: Best and worst prediction (measured by MSE) on the test set in terms of the RSAW.

6.2. Addition to the Branch Network

To try to improve the performance of the multi-shape model we propose a first extension: adding knowledge of the initial wound shape as an input to the branch network. Section 6.2.1 introduces this new architecture, while Section 6.2.2 considers the performance.

6.2.1. Architecture, Datasets, and Training

We hypothesised that the lack of information about the initial size and shape of the wound is the reason the previous DeepONet under-performs. That is why we propose to add it in the form of the quartet $(y_{cut}, x_m, y_m, x_{cut})$ to the branch network. Figure 6.1 visualises the locations of these points, which uniquely define the shape and size of the initial wound. For the point (x_m, y_m) we use the following strategy:

- If the initial wound shape is a rectangle, we take the corner point.
- If it is an ellipse or rhombus, we take a point that is roughly, but not necessarily exactly, in the middle of the wound boundary. This is already enough to uniquely define the shape, given that only these three shapes are present.

The new architecture we thus obtain is depicted in Figure 6.6.

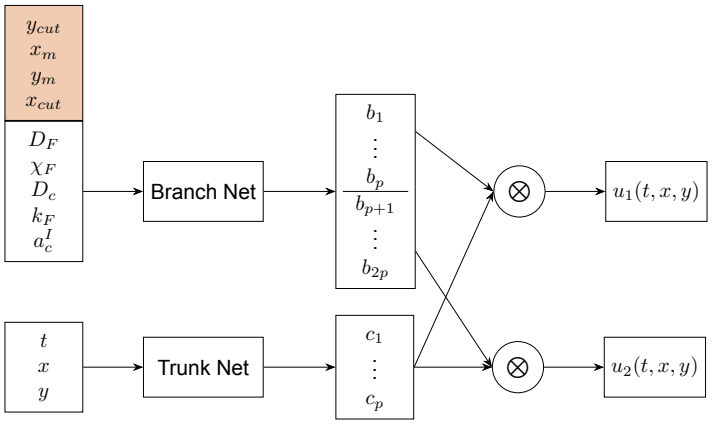


Figure 6.6: The DeepONet architecture when initial wound shape information is fed to the branch. The coloured block indicates an addition.

The training set is the same as before, but with the addition of the extra four inputs to the branch. We train the new network in the same way as in the previous case, using the settings described in Section 6.1.1. After training for 100 epochs, we find a training loss of 0.0052 and a validation loss of

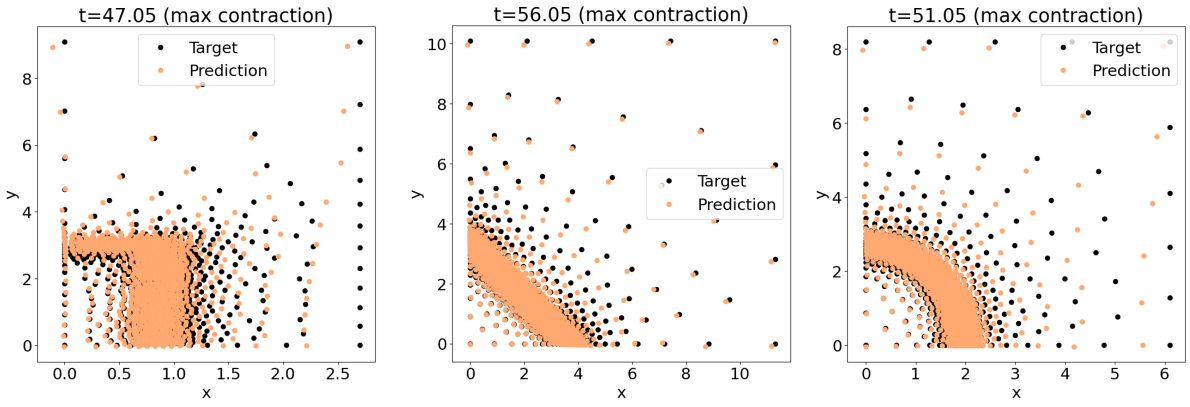
0.0035. The progression of the losses is very similar to Figure 6.2 and again we observe a stagnation of the MSE loss around 10^{-2}. This is already an indication of not much improvement.

6.2.2. Performance on the Test Set

To substantiate our suspicions, we examine the performance of the new model on the test set. Figure 6.7 depicts predictions vs. targets for the three samples considered in Figure 6.3. Note that in the former we only show the respective times of maximal contraction. This is enough to denote that there is no improvement compared to the latter. The same can be concluded from the scatter plots in Figure 6.8, which are almost indistinguishable from those in Figure 6.4. The error plots in Figure B.4 additionally reveal that the absolute errors over the domain have not decreased in magnitude.

Table 6.2: Performance of the DeepONet with the addition of initial wound shape info to the branch.

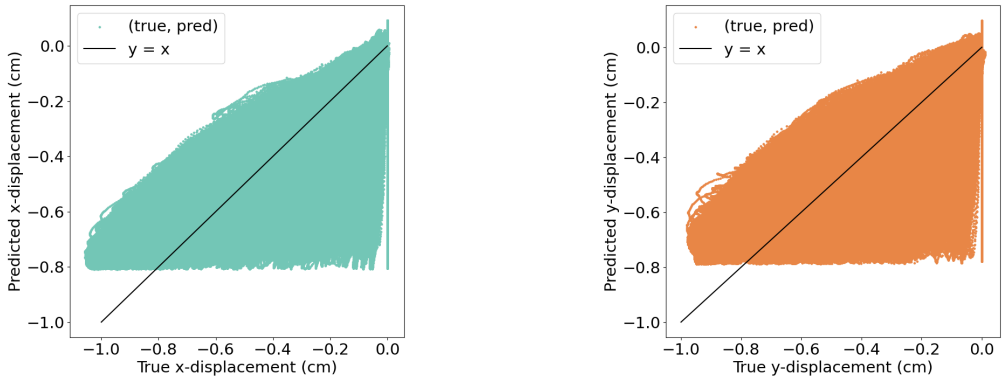
Performance measure	Result on test set
R^2	0.89762
aRRMSE	0.31995
aRelErr	0.12827



(a) Rectangular-shaped wound **(b)** Rhombus-shaped wound **(c)** Elliptic-shaped wound

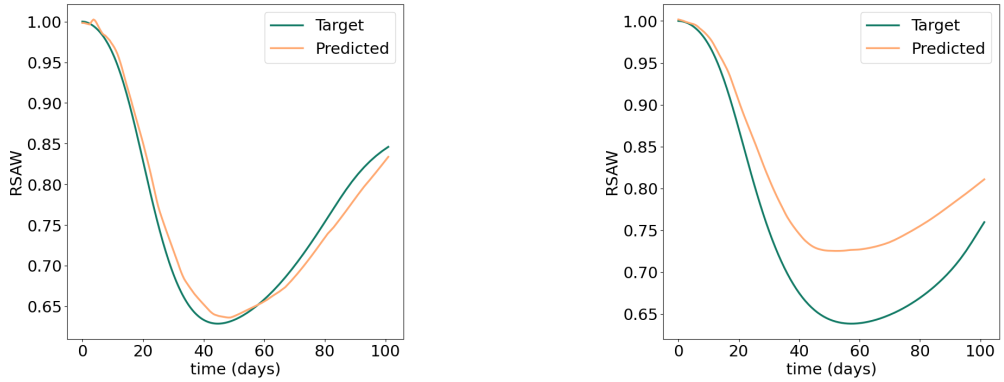
Figure 6.7: Prediction vs. target for three different samples from the test set, at their respective time of maximal contraction.

The performance metrics in Table 6.2 reflect a very slight increase in performance compared to Table 6.1. The R^2 score with 0.8976 is 0.7% higher than in the case where there is no info of the initial wound shape to the branch. The aRRMSE and aRelErr are 3% and 5.5% lower, respectively. This is not a significant improvement and we conclude that the updated DeepONet is still not able to accurately predict the displacements, despite the initial shape info to the branch. The same is confirmed in Figure 6.9, where we evaluate the model on the wound boundary to compute the RSAW over time. We observe that the best and worst predictions are very similar to the ones in Figure 6.5.



(a) x-direction **(b)** y-direction

Figure 6.8: True vs. predicted displacement in x- and y-direction on the test set.



(a) Best prediction **(b)** Worst prediction

Figure 6.9: Best and worst prediction (measured by MSE) on the test set in terms of the RSAW.

6.3. Addition to the Trunk Network

A natural second attempt is to include the information about the initial wound size and shape to the trunk network, as opposed to the branch. This section explores the new option, with Section 6.3.1 detailing the new architecture and training, and Section 6.3.2 evaluating the performance on the test set.

6.3.1. Architecture, Datasets, and Training

The updated DeepONet architecture we consider is illustrated in Figure 6.10. Here, the quartet $(y_{cut}, x_m, y_m, x_{cut})$ is taken as additional input to the trunk network. Besides that, the training and test set remain the same, as well as the training setup.

We train the model for 100 epochs, after which we obtain a training and validation loss of 0.0001. Figure 6.11 plots the MSE loss against the number of epochs. We observe that after approximately 60 epochs, both losses level off around 10^{-3}. Training for more epochs does not significantly decrease the loss further. Compared to Figure 6.2, a loss of two orders of magnitude lower is reached in the current case. This is already an indication that with our current setup we might have achieved the desired performance increase.

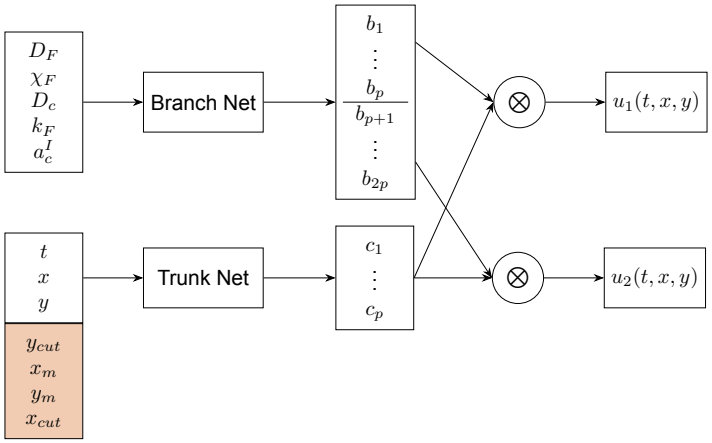


Figure 6.10: The DeepONet architecture when initial wound shape information is fed to the trunk. The coloured block indicates an addition.

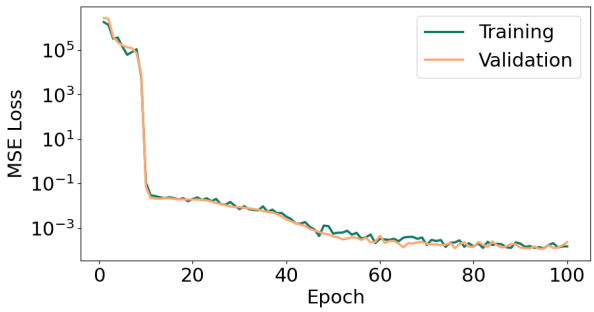
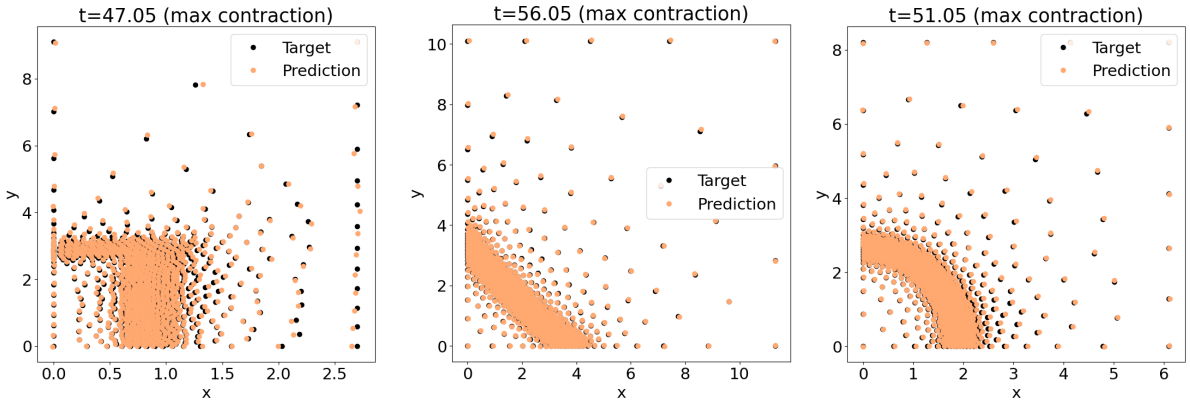


Figure 6.11: Training and validation losses.

6.3.2. Performance on the Test Set

We evaluate the current DeepONet on the test set. Figure 6.12 presents targets vs. predictions for three different samples in the test set at their respective time of maximal contraction. We may directly compare this figure to Figure 6.3 and Figure 6.7. Indeed, we see a very significant improvement in the current case. At and surrounding the wound boundary, the predictions mostly overlap with the targets. Even further from the wound, the predictions remain sufficiently accurate. Notably, especially for the rectangular-shaped wound with the smallest domain size, at the domain boundaries the predictions are still inaccurate. We even observe positive displacements there, which should not occur. Nevertheless, it seems that including the initial size and shape of the wound to the trunk network was the right choice.

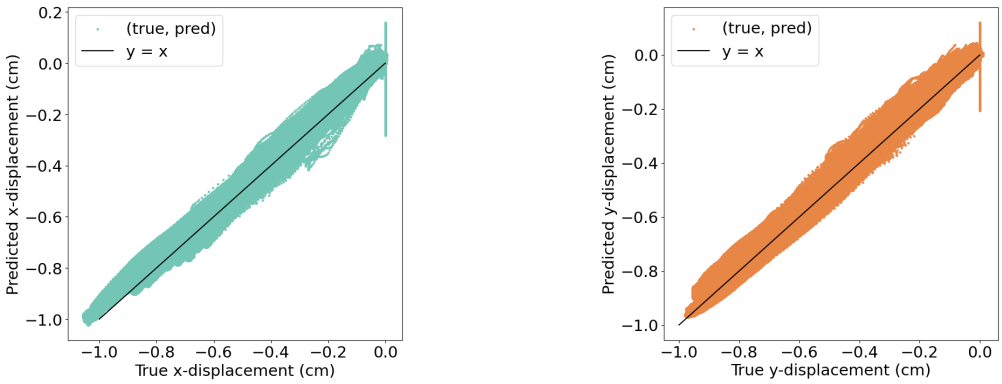


(a) Rectangular-shaped wound **(b)** Rhombus-shaped wound **(c)** Elliptic-shaped wound

Figure 6.12: Prediction vs. target for three different samples from the test set, at their respective time of maximal contraction.

This becomes even more apparent if we consider the scatter plots in Figure 6.13. Compared to both Figure 6.4 and Figure 6.8, we observe a major improvement. There is now a strong concentration around the lines $y = x$. This indicates that the model has now learned to accurately predict the x- and y-displacement over time. One notable outlier is the horizontal lines at $x = 0$: the true displacement should be zero, but the model still predicts (even positive) displacements. This occurs at the boundaries of the domain and is in accordance with what we have observed in Figure 6.12. Ideally, we would like to improve upon this as well.

More confirmation that providing knowledge about the shape and size of the wound to the trunk, rather than to the branch, enhances performance can be found in Table 6.3. Compared to Table 6.2, we now find an R^2 score of 0.9932, which is an increase of 10.6%. The aRRMSE is with its value of 0.0826 an impressive 74.2% lower, and the aRelErr (now 0.0451) is almost 65% lower than in the case of addition to the branch.



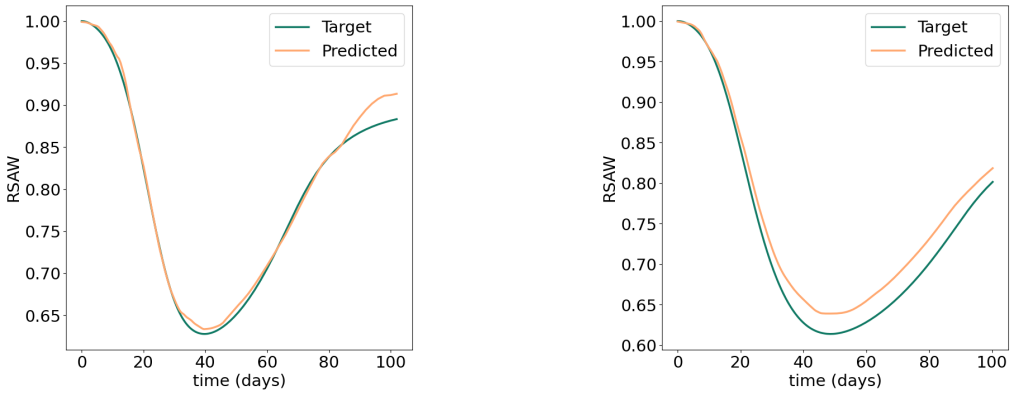
(a) x-direction **(b)** y-direction

Figure 6.13: True vs. predicted displacement in x- and y-direction on the test set.

Table 6.3: Performance of the DeepONet with the addition of initial wound shape info to the trunk.

Performance measure	Result on test set
R^2	0.99318
aRRMSE	0.08257
aRelErr	0.04506

Furthermore, we evaluate the trained model on the wound boundaries of our test set. Figure 6.14 provides the best and worst predictions in terms of the RSAW. Comparing this to the best and worst predictions in Figure 6.9, we see great improvements as well, which was to be expected. The best-case scenario is now a prediction that mostly overlaps with the target, with a small under-prediction of the maximal contraction of 0.9%. We do observe a larger overestimation of 3.4% of the final value at $t = 100$. In the worst-case scenario, the model overestimates the RSAW after $t = 18$ days. It is much less severe than what we have seen before: the minimum RSAW and the final value are respectively 4% and 2% higher than the target.



(a) Best prediction

(b) Worst prediction

Figure 6.14: Best and worst prediction (measured by MSE) on the test set in terms of the RSAW.

We conclude that the addition of the initial shape info to the trunk network has the desired effect we aimed for. The reason why addition to the branch had no effect could be that the trunk network's output was not sufficiently expressive. We know that the trunk maps the input coordinates (t, x, y) to a vector that represents the basis functions' values at that point. This is key for capturing the structure of the solution space of our PDE. It is to be expected that adding extra information to the trunk directly influences how the basis functions are learned and represented. Improved representation of basis functions can lead to a more accurate approximation of the solution, as we have seen here. If the trunk network is not expressive enough, adding this information to the branch might not help much. The coefficients generated by the branch are only as good as the basis functions they are being combined with. Addition to the trunk is also the most logical choice, because the initial shape info is more closely related to the spatial coordinates, rather than to the parameters that are input to the branch.

6.4. Sine Augmentation

We now have a multi-shape model with desired performance. Our last update aims to improve the accuracy of predictions at the domain boundaries. To this end, we introduce an additional block in the architecture called sine augmentation. This is explained in greater detail in Section 6.4.1. Lastly, Section 6.4.2 investigates the performance of our final model.

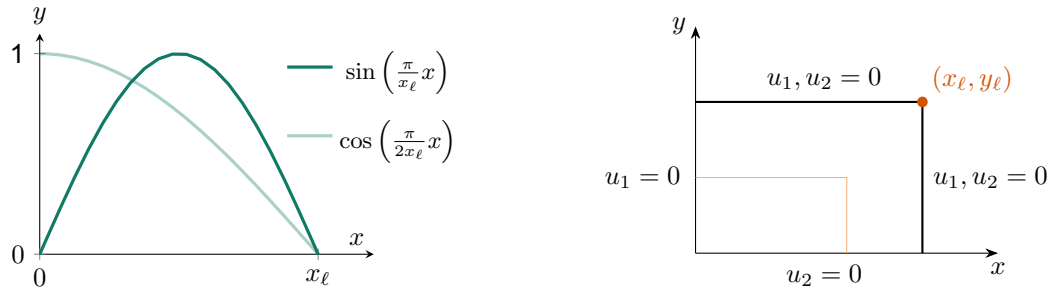
6.4.1. Architecture, Datasets, and Training

From the numerical simulations we know that the x- and y-displacements at the top and right boundaries of the domain are zero. Furthermore, we know that on the left boundary we have zero x-displacement, whereas on the bottom boundary we have zero y-displacement. We want to utilise this knowledge to improve the predictions at the four boundaries. The idea is to multiply the DeepONet outputs with functions that vanish at the boundaries, but remain close to one in the interior of the domain. If we rename the current outputs of the DeepONet $\hat{u}_1$ and $\hat{u}_2$, we propose the following:

$$u_1 = \hat{u}_1 \cdot \sin\left(\frac{\pi}{x_\ell} x\right) \cos\left(\frac{\pi}{2y_\ell} y\right), \tag{6.1}$$

$$u_2 = \hat{u}_2 \cdot \sin\left(\frac{\pi}{y_\ell} y\right) \cos\left(\frac{\pi}{2x_\ell} x\right). \tag{6.2}$$

Figure 6.15a gives a visualisation of the sine in Equation (6.1) and the cosine in Equation (6.2). We see that the sine vanishes in 0 and x_ℓ, which ensures that $u_1 = 0$ on the left and right boundaries of the domain. The cosine vanishes in x_ℓ only. This ensures that $u_2 = 0$ on the right boundary as well. Similarly, the cosine in Equation (6.1) and the sine in Equation (6.2) ensure that $u_1, u_2 = 0$ on the top boundary and $u_2 = 0$ on the bottom boundary. Figure 6.15b summarises the effect on all four boundaries.



(a) The functions used (in x).

(b) Result on the domain boundaries.

Figure 6.15: Visualisation of the sine augmentation.

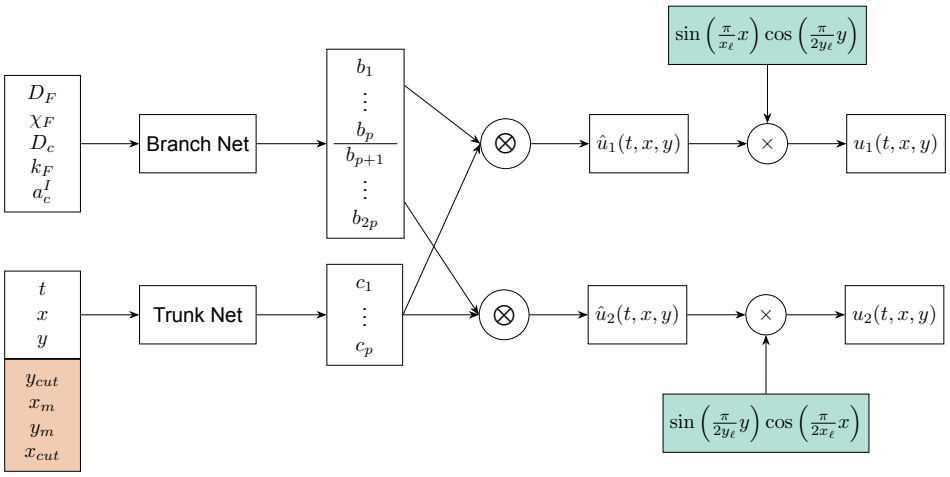


Figure 6.16: The final DeepONet architecture considered for predicting dermal displacement. The initial wound shape info $(y_{cut}, x_m, y_m, x_{cut})$ is fed to the trunk. Additionally, sine augmentation (green blocks) is applied, making use of the domain size (x_ℓ, y_ℓ).

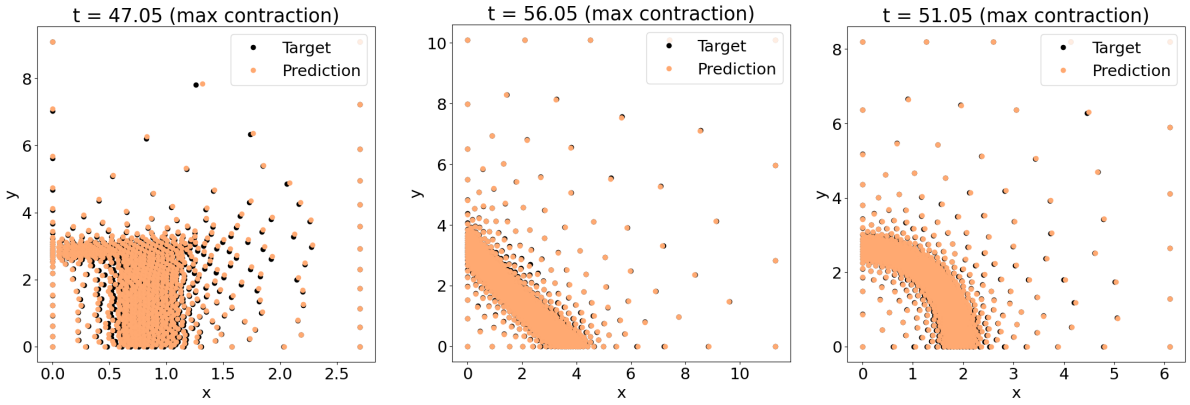
The updated DeepONet architecture with the addition of the sine augmentation blocks is visualised in Figure 6.16. We train the network using the same training set as before. The only difference is that for each training sample, the sine augmentation step requires additional information in the form of the point (x_ℓ, y_ℓ). That is why, each finite element simulation now contributes the following quartet to the training data:

$$\left[\begin{bmatrix} D_F, \chi_F, D_c, k_F, a_c \\ D_F, \chi_F, D_c, k_F, a_c \\ \vdots \\ D_F, \chi_F, D_c, k_F, a_c \end{bmatrix}, \begin{bmatrix} t_1, x_1, y_1, y_{cut}, x_m, y_m, x_{cut} \\ \vdots \\ t_1, x_{20}, y_{20}, y_{cut}, x_m, y_m, x_{cut} \\ \vdots \\ t_{10}, \hat{x}_1, \hat{y}_1, y_{cut}, x_m, y_m, x_{cut} \\ \vdots \\ t_{10}, \hat{x}_{20}, \hat{y}_{20}, y_{cut}, x_m, y_m, x_{cut} \end{bmatrix}, \begin{bmatrix} u_1(t_1, x_1, y_1), u_2(t_1, x_1, y_1) \\ \vdots \\ u_1(t_1, x_{20}, y_{20}), u_2(t_1, x_{20}, y_{20}) \\ \vdots \\ u_1(t_{10}, \hat{x}_1, \hat{y}_1), u_2(t_{10}, \hat{x}_1, \hat{y}_1) \\ \vdots \\ u_1(t_{10}, \hat{x}_{20}, \hat{y}_{20}), u_2(t_{10}, \hat{x}_{20}, \hat{y}_{20}) \end{bmatrix}, \begin{bmatrix} x_\ell, y_\ell \\ x_\ell, y_\ell \\ \vdots \\ x_\ell, y_\ell \end{bmatrix}\right].$$

The first column is input to the branch, the second column is input to the trunk, the third column represents the target values, and the final column evaluates the sines and cosines in the sine augmentation step. We train the model with the same settings for 100 epochs. The progression of the losses is very similar to Figure 6.11, again yielding a training and validation loss of 10^{-4}. This already indicates that adding the sine augmentation blocks did not negatively impact the performance.

6.4.2. Performance on the Test Set

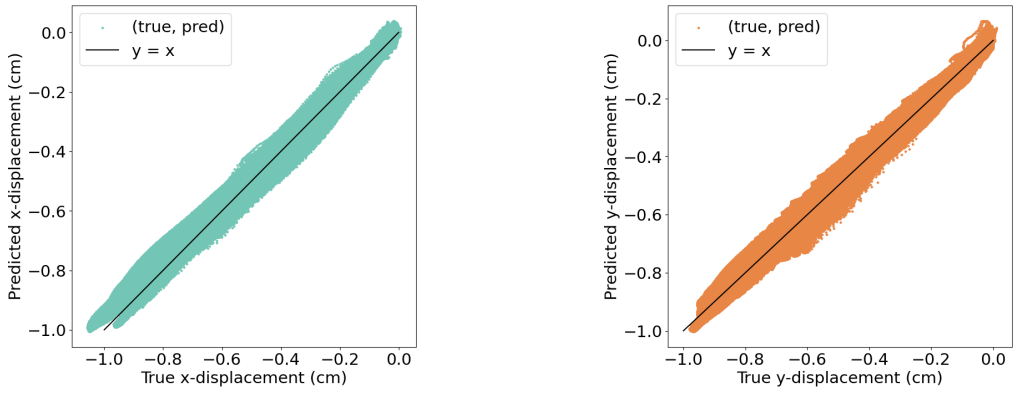
To be certain, we examine the performance on our test set. The risk with incorporating the sine augmentation is that although the predictions at the boundaries improve, it negatively impacts the predictions in the interior of the domain. This seems not to be the case if we consider the three examples in Figure 6.17. Comparing the predictions vs. targets there to those in Figure 6.12, we observe very similar results. It is immediately apparent from the predictions at the domain boundaries that the sine augmentation has the desired effect (best visible in Figure 6.17a and Figure B.6). In the former figure we do notice that surrounding the wound boundary, the prediction is less accurate in some points. This is not the case in Figure 6.17c, where the prediction seems to have improved in almost all points.



(a) Rectangular-shaped wound **(b)** Rhombus-shaped wound **(c)** Elliptic-shaped wound

Figure 6.17: Prediction vs. target for three different samples from the test set, at their respective time of maximal contraction.

To obtain a more conclusive answer, we turn to the scatter plots in Figure 6.18 and the performance metrics in Table 6.4. In the former we again observe a strong concentration of points around the lines $y = x$. Compared to Figure 6.13, we notice that the outliers at $x = 0$ are gone. This can be ascribed to the effect of the sine augmentation. We compute an R^2 score of 0.996, which is a slight increase of 0.2% compared to the previous case where there was no sine augmentation (Table 6.3). The aRRMSE is 0.0633, which is a 23.4% improvement, while the aRelErr (now 0.0315) has the largest decrease of 30.1%.



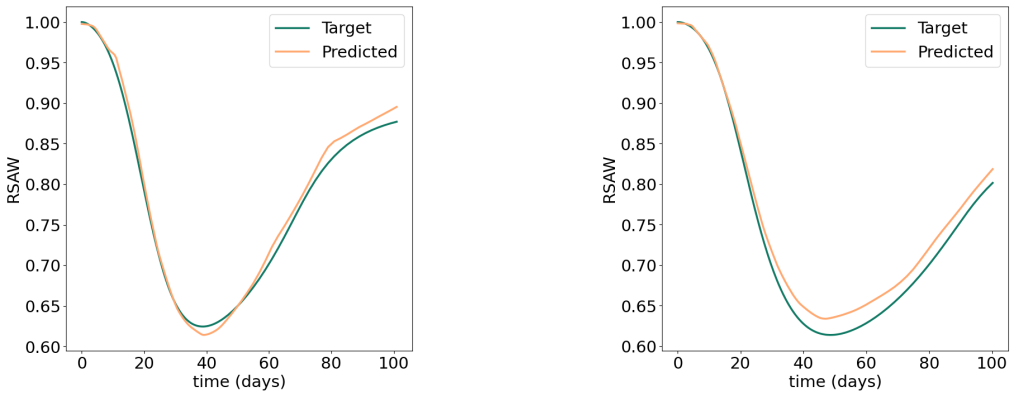
(a) x-direction **(b)** y-direction

Figure 6.18: True vs. predicted displacement in x- and y-direction on the test set.

Table 6.4: Performance of the DeepONet with the addition of initial wound shape info to the trunk and sine augmentation.

Performance measure	Result on test set
R^2	0.99598
aRRMSE	0.06326
aRelErr	0.03149

Evaluating the DeepONet on the wound boundary produces the RSAW curves in Figure 6.19. We find that the worst prediction is indistinguishable from the worst prediction in Figure 6.14. The current best prediction gives an underestimation of the minimum RSAW of 1.7%. The predicted final value at $t = 100$ is 2% higher than the target, which is a slight improvement compared to Figure 6.14a. From this we can conclude that the addition of the sine augmentation has not decreased the prediction accuracy on the wound boundary. Compared with our previous findings, we conclude that it has realised an overall performance improvement.



(a) Best prediction

(b) Worst prediction

Figure 6.19: Best and worst prediction (measured by MSE) on the test set in terms of the RSAW.

6.5. Conclusion

At the start of this chapter, we asked whether it is possible to train a DeepONet on multiple wound shapes, to accurately predict the dermal displacement across the entire domain over time. We commenced the investigation by applying our initial DeepONet architecture (Figure 5.2) to a training set containing data derived from multiple initial wound shapes (a square, a rhombus, and an ellipse) with varying sizes. This proved to be inadequate, as evidenced by the results in Section 6.1.2. The model struggled to learn the correct patterns in the solutions, resulting in a lacking performance.

As a first attempt to remedy this, we considered an updated DeepONet architecture (Figure 6.6), incorporating knowledge of the initial wound shape and size as extra input to the branch. However, as indicated in Section 6.2.2, this was not much of an improvement. The second attempt proved to be more effective: feeding these extra inputs to the trunk (Figure 6.10) resulted in a significant performance increase. In Section 6.3.2 we saw that the network was then able to accurately predict the displacements over time. Nevertheless, displacement at the domain boundaries remained problematic. To address this, we introduced a final version of the architecture, with the addition of sine augmentation blocks (Figure 6.16). Evaluation results in Section 6.4.2 confirmed that this final version was best performing and met our requirements.

We now have successfully constructed and trained a multi-shape model that predicts dermal displacement over time with satisfactory accuracy. The subsequent step is to evaluate its generalisation capabilities.

7

Predictions on Convex Combinations of Wound Shapes

This chapter serves the purpose of answering our final sub-question: can we train a DeepONet on multiple wound shapes, to accurately predict the dermal displacement across the entire domain over time, that *generalises well to convex combinations of the basic shapes*? Section 7.1 introduces a new test set containing convex combinations of the shapes we used previously and evaluates the performance of the DeepONet on this new dataset. In Section 7.2 we investigate extending the predictions to one year, as opposed to 100 days. Lastly, Section 7.3 formulates a conclusion.

7.1. Convex Test Set

For the remainder of this chapter, we consider the final DeepONet architecture to be fixed (Figure 6.16 or the final model in Figure 7.5) and no new changes will be introduced. We aim to assess the performance of the model on what we call the *convex test set*. This is a more generic dataset, containing convex combinations of the three basic shapes used for training. Section 7.1.1 gives a description of the dataset and in Section 7.1.2 we put it to the test. Lastly, Section 7.1.3 compares the performance of the previous architectures when evaluated on the convex test set.

7.1.1. Description

Let geo_i with $i \in \{1, 2, 3\}$ represent three instances of the different initial wound geometries (a square, a rhombus, and an ellipse). Then, a convex combination of the three shapes can be written as

$$\sum_{i=1}^{3} \alpha_i geo_i, \tag{7.1}$$

where $\alpha_i \in \mathbb{R}$, satisfying $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. Figure 7.1 provides a visualisation.

The convex test set it created by executing 150 finite element simulations. For each simulation, we create a convex combination of the three basic shapes. To this end, we first fix the sizes of the latter by uniformly choosing $x_{cut}, y_{cut} \in (0, 5)$ centimeters. Then, we randomly select three weights that sum up to one and perform the multiplication as given in Equation (7.1).

The remainder of the data sampling strategy is similar to the one used for generating the training data described in Section 5.3. The only difference now is that for each finite element run, we sample all time steps and for each time step, we sample all spatial coordinates in the domain. This procedure results in a convex test set containing 18.035.821 data points. We note that the quartet $(y_{cut}, x_m, y_m, x_{cut})$ no longer uniquely defines the initial wound shape for this dataset.

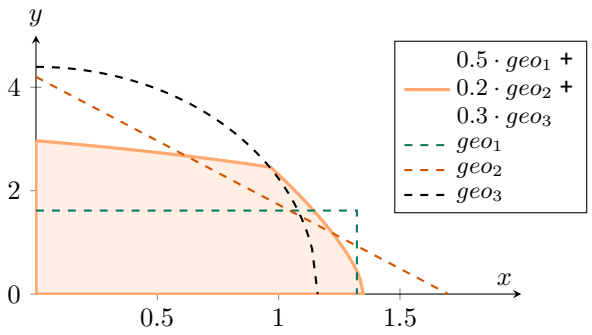


Figure 7.1: Example of a convex combination of the three basic wound shapes.

7.1.2. Performance

We evaluate the DeepONet, which is trained on the three basic wound shapes, on the convex test set. Figure 7.2 presents an example of the obtained predictions versus the corresponding targets at $t = 0$, at the time of maximal contraction, and at $t = 100$. We observe very accurate predictions, both at the wound boundary and further away. At $t = 0$, the displacements are zero everywhere, which is correctly predicted. It seems that the prediction accuracy is better at $t = 100$ than at the time of maximal contraction. Nonetheless, given that our model is trained on the three basic wound shapes, Figure 7.2 gives an indication of good generalisation abilities. If we compare the middle figure to Figure 6.17, we cannot say that we observe a performance decrease compared to the original test set.

Further validation can be found in the scatter plots in Figure 7.3, where the true displacements are plotted against the predicted displacements. We observe a strong concentration around the lines $y = x$ with no unusual outliers, very similar to Figure 6.18. This indicates that our model gives good predictions and that it generalises well to convex combinations of the shapes is was trained on. We note that the model is slightly better at predicting the x-displacement. For the y-displacement, the model appears to be less accurate for displacements between -0.7 and -0.4 cm, as indicated by the thicker band around the line $y = x$ in this range. For both x- and y-displacement, we observe that the model is best at predicting very small magnitude displacements (close to zero) and the largest magnitude displacements (around -0.8 cm). The former occurs near the boundary of the domain at all time steps, and at all spatial coordinates when t is small. The latter occurs at the wound boundary at maximal contraction, for the largest wound sizes present in the convex set.

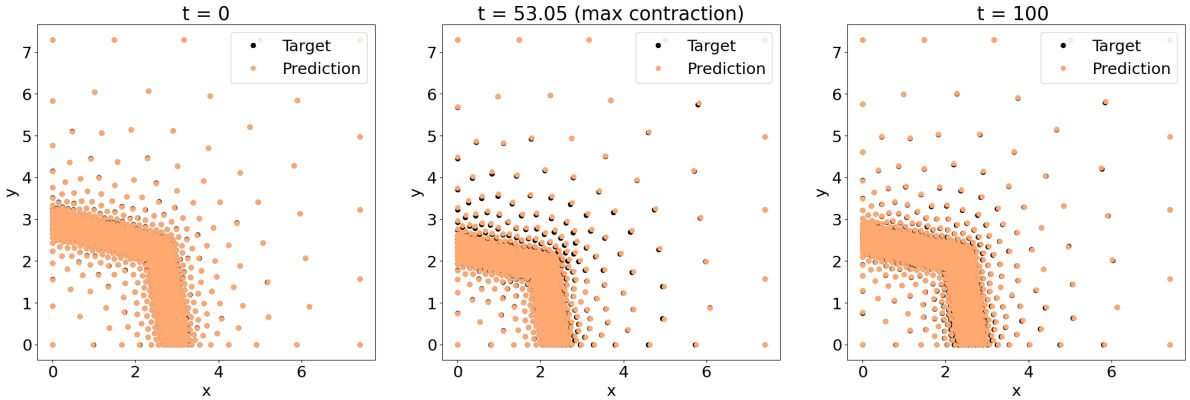
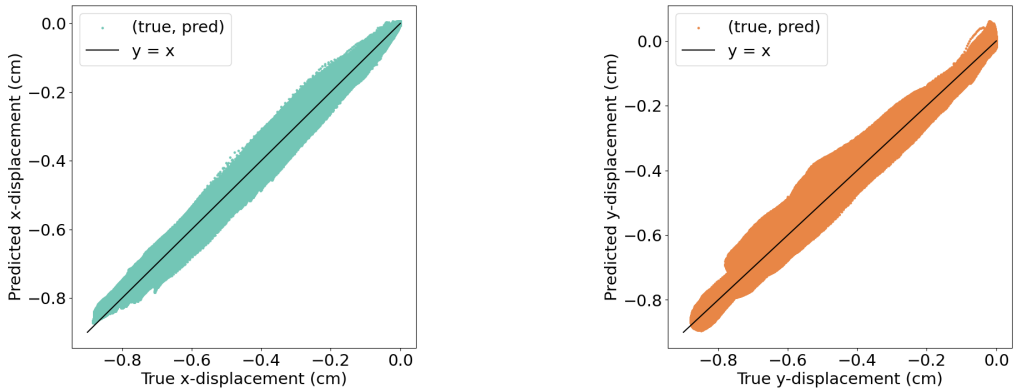


Figure 7.2: Prediction vs. target for one samples from the convex test set at $t = 0$, at the time of maximal contraction, and at $t = 100$.

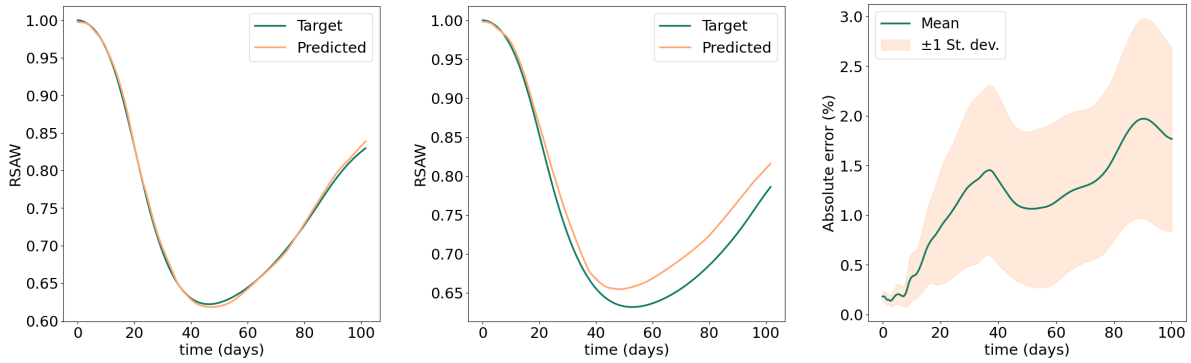


(a) x-direction **(b)** y-direction

Figure 7.3: True vs. predicted displacement in x-and y-direction on the convex test set.

Additionally, we evaluate our trained model on the wound boundary. Figure 7.4a and Figure 7.4b show the best and worst predictions in terms of the RSAW on the convex set. In the best case, the RSAW curve mostly overlaps the target, with a very slight underestimation of the maximal contraction and overestimation of the final value at $t = 100$ days. This result outperforms the best prediction on the original test set containing the three basic shapes (Figure 6.19a). In the worst case, the model predicts an earlier maximal contraction (49 days vs. 53) and retraction, underestimating the RSAW by approximately 4% from $t = 55$ days and onwards. This is slightly worse than the worst prediction in Figure 6.19b.

Figure 7.4c gives more insight in the behavior of the error over time. Here, the mean and standard deviation of the absolute error over the convex test set are depicted. We observe that the absolute error increases over time. The model is most accurate in the first 18 days, where the error is up to 0.5% on average. After this time, the error, and the uncertainty thereof, start to increase. This coincides with very steep contraction. A notable peak can be observed around day 38, after which there is a local minimum (of 1.2% on average) around day 50. The latter coincides with the average time of maximal contraction, indicating that the model is better at predicting the minimum RSAW (maximal displacement). The highest absolute error, with the largest uncertainty, is around day 90. For all times, the standard deviation of the error never exceeds 3%. This is confirmed in the absolute error plots over the domain in Figure B.7. Since the model predicts displacements within the rough range of $[-0.9, 0]$ centimeters, a 3% error corresponds to deviations on the order of tenths of millimeters.



(a) Best prediction **(b)** Worst prediction **(c)** Error behaviour

Figure 7.4: Best and worst prediction on the convex test set in terms of the RSAW. Mean and standard deviation of absolute error as a function of time. The mean is taken over all 150 samples in the convex test set.

7.1.3. Comparison to Different Architectures

In Chapter 6, we compared the performances of different DeepONet architectures on the test set containing the three basic wound shapes. As a verification, we perform the same comparison on the

convex test set. Figure 7.5 summarises the five different settings taken into consideration. Note that case 4, where we only have the addition of the sine augmentation, is the only one not examined in Chapter 6. We include it here to observe the effect in isolation.

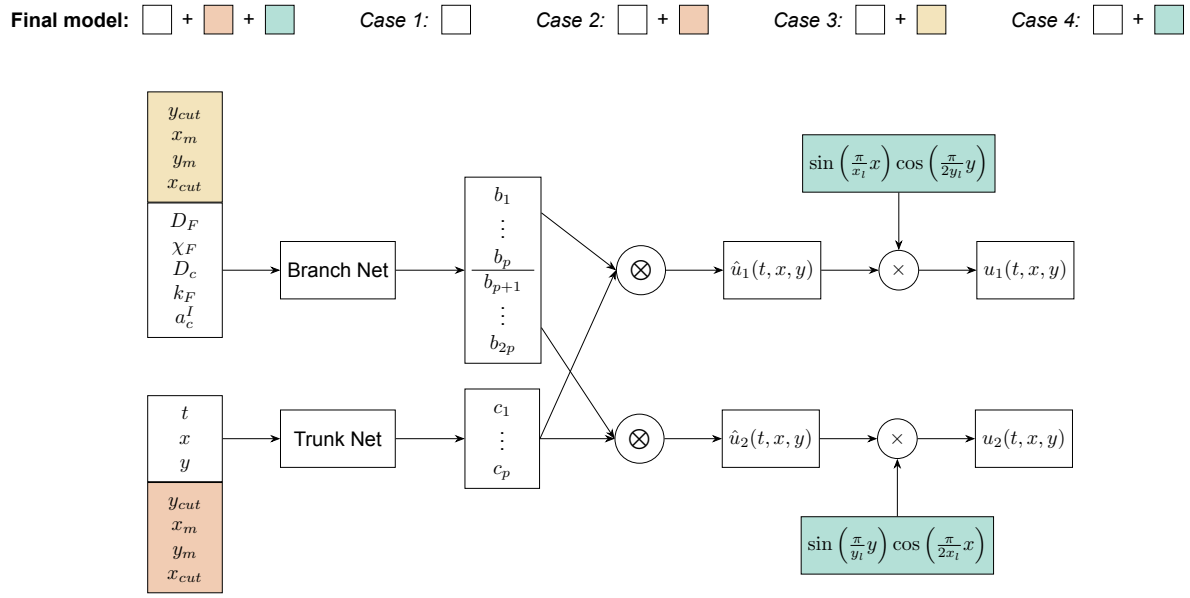


Figure 7.5: Summary of the DeepONet architectures used for predicting dermal displacement. Case 4 is the only one not considered in Chapter 6.

Table 7.1: Performance of the final DeepONet on the convex test set compared to four other setups (please refer to Figure 7.5). Italics indicate best performance.

Performance metric	**No sine aug. and no initial shape info** (Case 1)	**Initial shape info to branch** (Case 2)	**Initial shape info to trunk** (Case 3)	**Sine augmentation** (Case 4)	**Sine aug. and initial shape info to trunk** (Final model)
R^2	0.90040	0.89613	0.99155	0.89083	*0.99437*
aRRMSE	0.31560	0.32229	0.09166	0.33032	*0.07497*
aRelErr	0.12160	0.11673	0.04432	0.15050	*0.03429*

Table 7.1 presents the computed performance metrics for the different cases. Each column corresponds to an architecture in Figure 7.5. Columns one, two, three, and five can be compared to Table 6.1, 6.2, 6.3, and 6.4, respectively. The latter contain the performance metrics for the same architectures, but evaluated on the original test set. We observe that in each case, the performance on the convex test set is slightly worse than on the test set containing the three basic shapes. This is to be expected, as the network has never seen convex combinations of the shapes during training. However, the difference is not that significant. For example, if we consider the final model, we find $R^2 = 0.9944$, aRRMSE $= 0.075$ and aRelErr $= 0.0343$. Comparing this to Table 6.4, we find a decrease of 0.16%, an increase of 18.5% and an increase of 8.9%, respectively. This is minor compared to the performance differences observed when transitioning from one architecture to another. From these values, we conclude that the final model can excellently reproduce the finite element simulations and is able to generalise to convex combinations.

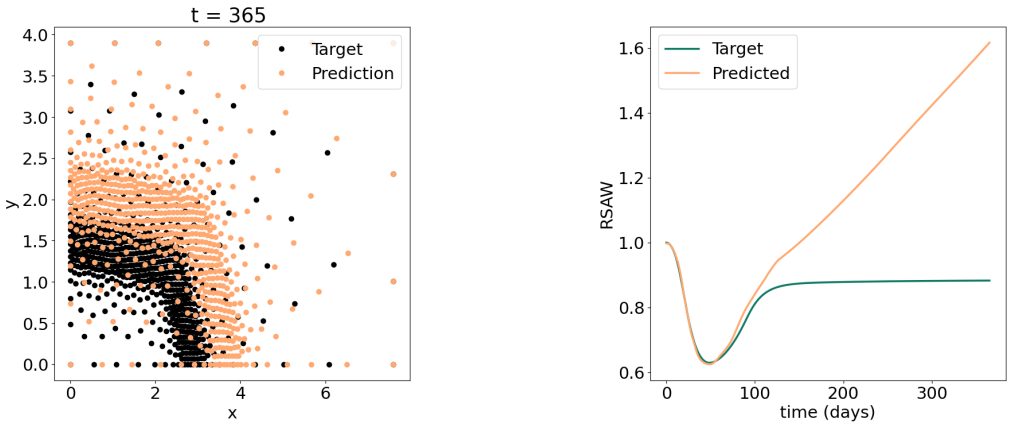
Comparing the different cases in Table 7.1, we observe similar trends to those found in Chapter 6. The 'skeleton' DeepONet, where the model has no information about the initial wound shape and no sine augmentation is applied (case 1), exhibits the worst performance. We find $R^2 = 0.9004$, aRRMSE $= 0.3156$ and aRelErr $= 0.1216$. Adding initial wound shape info in the form of the quartet $(y_{cut}, x_m, y_m, x_{cut})$ to the branch (case 2), even results in slightly worse performance when evaluated on the convex test set. The most significant improvement occurs when this initial shape info is added

to the trunk (case 3), leading to a decrease of 71% in the aRRMSE and a decrease of 63.6% in the aRelErr, compared to case 1. This again suggests that the initial shape info should be input to the trunk. Moreover, it demonstrates that even though $(y_{cut}, x_m, y_m, x_{cut})$ can no longer uniquely define the initial wound shape in the case of convex combinations, it still provides sufficient information for a good prediction. Case 4, where the skeleton DeepONet only has the addition of the sine augmentation blocks, is the worst performing one with $R^2 = 0.8908$, aRRMSE $= 0.3303$, and aRelErr $= 0.1505$. From this we conclude that the network cannot sufficiently learn the dermal displacements without knowledge of the initial shape of the wound. Additionally, sine augmentation provides an improvement in performance only when the model already performs sufficiently without it. Indeed, when combining case 3 and 4, which forms the final model, we observe that it achieves the best overall performance, which proves to hold for convex combinations as well.

7.2. One Year Prediction

Until now, we have taken $t \in [0, 100]$ days for all predictions to save time on numerical computation. However, the finite element model can simulate the solution for an entire year. We are interested to see if our final DeepONet is capable of predicting $t \in [0, 365]$ days as well. As an initial trial, we consider the best prediction in terms of the RSAW, evaluated on the convex test set (Figure 7.4a). Taking the corresponding parameter values and settings, we re-simulate the finite element model, now with $t_{end} = 365$. We evaluate the DeepONet, which is trained on $t \in [0, 100]$, on one year. A comparison between the two is made in Figure 7.6. We observe that at $t = 365$, the network predicts a wound that is much more detracted than the target. This is particularly apparent in Figure 7.6b, where we see that the predicted RSAW is linearly increasing after $t = 100$ days and no asymptotic value is reached. This means the network predicts a wound that continues to grow in size, becoming 1.6 times larger after a year than it initially was. Of course, this behaviour is completely incorrect. It is not surprising, since the network has never seen data beyond $t = 100$ during training. It correctly learned that the wound starts to detract after roughly 50 days, and it simply extrapolates this pattern to the entire year.

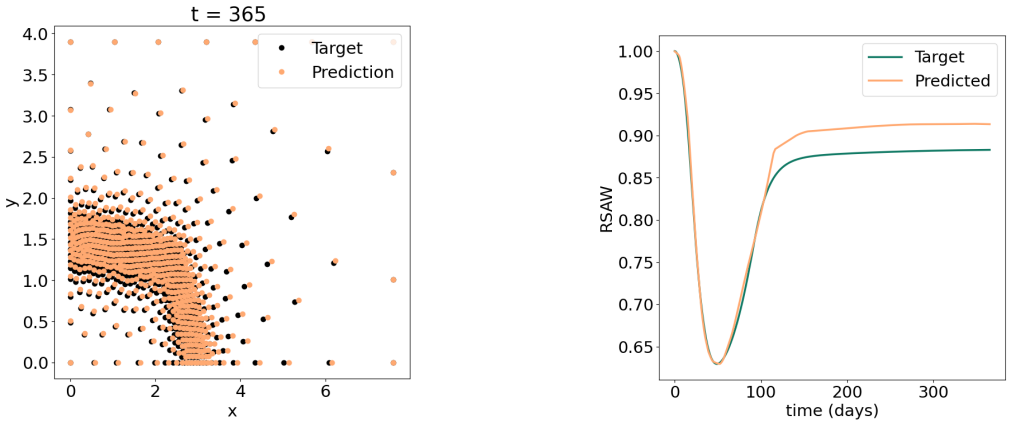
To remedy this, we try extending the training set with 50 new finite element simulations where $t_{end} = 365$. Note that the training data is comprised of the three basic wound shapes, so we add 50 new variations of these shapes. We now have 750 samples with $t_{end} = 100$ and an additional 50 with $t_{end} = 365$. We draw 10 time steps per finite element simulation, for the new data enforcing $t \in (100, 365]$, and 20 spatial coordinates per time step. We then re-train the model using this extended dataset. Figure 7.7 shows the improved predictions. We observe that the model has now learned the asymptotic behaviour of the RSAW and the predictions at $t = 365$, however, it still underpredicts the final contraction with 3.3%.



(a) Prediction vs. target at $t = 365$

(b) Corresponding RSAW curves

Figure 7.6: Result of evaluating the DeepONet (trained on $t \in [0, 100]$) on a whole year, for one sample from the convex test set.



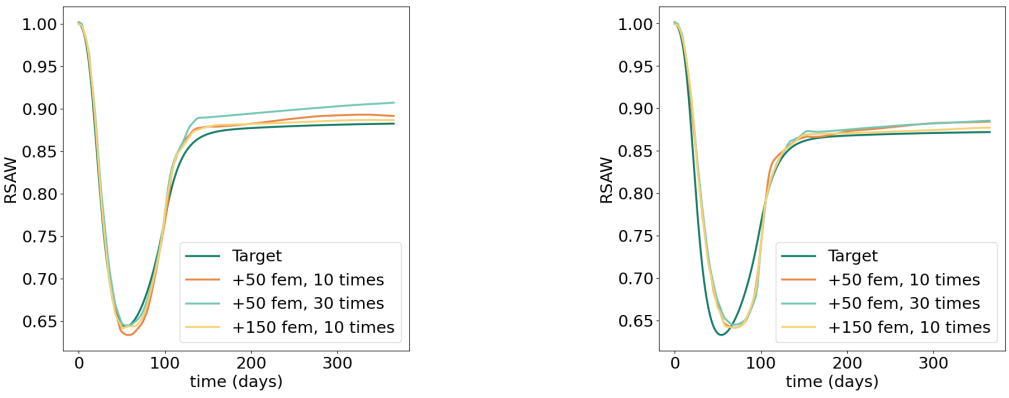
(a) Prediction vs. target at $t = 365$

(b) Corresponding RSAW curves

Figure 7.7: Result after adding only 50 extra samples with $t_{end} = 365$ to the training data.

To investigate whether we can improve the predictions for $t \in (100, 365]$, we experiment with different data sampling strategies. Specifically, we are interested to compare the effects of incorporating additional data from new finite element simulations, versus drawing more data points within existing samples. Practically, this comparison reflects the difference between including data of more burn patients versus collecting more data per patient. We generate a new convex test set for evaluation, existing of 50 finite element runs with $t_{end} = 365$. Figure 7.8 shows the results from three different training data sampling strategies. For each case, we initialise the network with the learned parameters from the final DeepONet trained on $t \in [0, 100]$, to facilitate easier learning of the correct solutions. In that way, we hope to embed prior knowledge, so that the network does not have to start learning from scratch.

Figure 7.8 demonstrates the effectiveness of the initialisation strategy. If we compare the RSAW curves for the scenario where 50 finite element simulations are added to the training set, with 10 time steps sampled in $t \in (100, 365]$ ('+50 fem, 10 times') to the prediction in Figure 7.7b, we observe a higher accuracy in the former. This improvement is despite both cases using the same additional data and training settings.



(a) Best prediction

(b) Worst prediction

Figure 7.8: Best and worst prediction in terms of the RSAW on a small convex test set, comparing different training data sampling strategies. The legend denotes the number of finite element simulations added to the training set, and the number of time steps sampled per simulation.

Furthermore, we compare the scenario where 50 finite element simulations are added to the training set, with 30 time steps sampled for each, versus the scenario where 150 runs are added, with 10 time steps sampled. Note that both result in the same number of additional data points (30.000). In both the best and worst cases, we observe that the second scenario is better performing. Sampling more data per existing finite element run does not lead to much improvement, in fact, in Figure 7.8a we even

observe a decrease compared to selecting 10 time steps. Adding new, unique data does result in a performance increase. In the best case, the predicted RSAW at $t = 365$ is now only 0.47% higher than the target. In the worst case, this is 0.6%.

The reason why increasing the number of additional data points by including new data appears to be more effective, could be that this increases the diversity of the training set more. It exposes the network to a wider variety of features, which can enable it to generalise better. On the other hand, introducing more data per existing samples can introduce redundancy, as the network is repeatedly exposed to the same patterns. Since these patterns were already learned in the original data, additional exposure does not result in significant improvement.

7.3. Conclusion

At the outset of this chapter, we asked the question whether our DeepONet, that is trained on multiple wound shapes to predict the dermal displacement across the entire domain over time, can generalise well to convex combinations of the basic shapes. To find out, we generated a new test set including finite element simulations based on convex combinations of the three basic wound shapes. The results in Section 7.1.2 demonstrate that the performance on this convex test set is much comparable to that on the original test set (containing the basic shapes). This indicates very good generalisation abilities, which implies we can answer our question in the affirmative. Furthermore, we compared the performance of different network architectures (Figure 7.5) on the convex test set, again concluding that our final model is best at reproducing the finite element simulations.

Lastly, we investigated extending DeepONet predictions to one year, rather than just 100 days. To this end, we expanded the training set with 50 new finite element simulations, where $t_{end} = 365$. Initialising the network with the learned parameters from the model trained on $t \in [0, 100]$ proved to be effective. We compared the RSAW curves for two scenarios: one where 50 finite element simulations were added to the training set, with 30 time steps sampled in $t \in (100, 365]$, and another where 150 runs were added, with 10 time steps each. We found that adding new, unique data (the first scenario) results in the greatest improvement. The may be because introducing new information leads to a more varied learning experience, whereas increasing the number of data points per existing samples tends to add redundancy, leading to less improvement.

We have now successfully trained a DeepONet on multiple initial wound shapes, demonstrating strong generalisation on convex combinations. Moreover, we have shown that by adding a limited number of samples from one-year simulations to the training set, the network achieved reasonable prediction of the RSAW over the entire year. It is to be expected that including more data with $t_{end} = 365$ could further enhance prediction accuracy.

8

Conclusion

This chapter presents the conclusions to this study. Sections 8.1 to 8.3 focus on predictions for single wound shapes, multiple wound shapes, and convex combinations of wound shapes, respectively. Finally, Section 8.4 answers our main research question.

8.1. Predictions on a Single Wound Shape

The first step in answering our main research question was to investigate the feasibility of training a neural operator based on the DeepONet architecture on one single wound shape. The aim was to accurately predict the dermal displacement across the entire domain over time. To this end, we trained a DeepONet with an architecture as illustrated in Figure 5.2, consisting of a branch and trunk network. The branch network takes as input five parameters from the morphoelastic model describing burn injuries, while the trunk takes as input a coordinate (t, x, y) in which we want to evaluate the solution. The outputs of these two networks are combined through a dot product, which produces the final DeepONet output. The network thus learns the solution operator that maps the parameter space (represented by the five parameters) to the displacement field (the x- and y-displacements at a given point (t, x, y)). For training, we used a dataset derived from a single rhombus-shaped initial wound of fixed size (Figure 5.1) and we fixed $t_{end} = 100$ days.

When evaluating the trained DeepONet on unseen test data, we found that it demonstrates very strong predictive capabilities in modelling the dermal displacement field. It achieved an R^2 score of 0.9975, an aRelErr of 0.025, and an aRRMSE of 0.0498, indicating that the model is highly accurate in reproducing the finite element simulations, with only minor errors. However, we observed that the model's accuracy seems to diminish slightly over time, where deviations become more noticeable at the boundaries of the domain. Specifically, positive displacements were predicted in areas where displacements should be zero, likely due to the fact that boundary conditions were not explicitly enforced. On the wound boundary, the model generally performs well. In the worst-case, it underestimates the maximal contraction by 2.5% and predicts a final RSAW at $t = 100$ that is 3% lower than the target. Nonetheless, this corresponds to deviations on the order of millimeters, which is sufficiently accurate for practical medical applications. We conclude that DeepONets can serve as accurate and fast surrogates for the finite element simulations on a single initial wound geometry with fixed size.

8.2. Predictions on Multiple Wound Shapes

The second step in addressing our main research question was to explore training a DeepONet on multiple wound shapes, to accurately predict the dermal displacement across the entire domain over time. We began by applying our initial DeepONet architecture (Figure 5.2) to a training set containing data derived from multiple initial wound shapes (a square, a rhombus, and an ellipse) with varying sizes. This approach proved to be inadequate, as evidenced by the performance metrics. The model achieved an R^2 score of 0.8911, an aRRMSE of 0.323, and an aRelErr of 0.1358, reflecting its struggle to learn the correct patterns in the solutions. We attributed the lacking performance to overfitting to

the largest wound shapes present in the training set and the network's inability to recognise smaller wounds, due to a lack of knowledge of the initial wound size and shape.

As a first attempt to address this issue, we considered an updated DeepONet architecture (Figure 6.6) that incorporated the initial wound shape and size as extra inputs to the branch network. Specifically, this information included x_{cut} and y_{cut}, which indicate the points where the initial wound boundary intersects the x- and y-axis, respectively, along with the point (x_m, y_m), approximately located at the centre of the wound boundary. These parameters collectively describe the initial wound size and shape in a unique way, given that the three shapes we consider are a rectangle, a rhombus, and an ellipse. However, upon evaluation of the trained model, we observed no significant improvement. We found an R^2 score of 0.8976, an aRRMSE of 0.312, and an aRelErr of 0.1283, leading us to conclude that the updated DeepONet still failed to accurately predict the displacements, despite the initial shape info to the branch.

The second attempt proved to be more effective: feeding the quartet $(x_{cut}, x_m, y_m, y_{cut})$ to the trunk network (Figure 6.10) resulted in a significant performance increase. The trained DeepONet now achieved an R^2 score of 0.9932, an aRelErr of 0.0451, and an aRRMSE of 0.0826. This is respectively an increase of 10.6%, a decrease of 65%, and a decrease of 74.2%, compared to the case of addition to the branch. The scatter plots where true displacements are plotted against the predicted displacements (Figure 6.13) showed a strong concentration around the line $y = x$, confirming that the model now accurately predicts the x- and y-displacements. However, some inaccuracies still remained at the domain boundaries, where zero displacements are incorrectly predicted. Evaluation on the wound boundary also demonstrated significant improvements, with minimal underpredictions and overestimations of the RSAW over time. The results confirm that adding initial wound shape and size information to the trunk network effectively enhances model accuracy, likely because it enables a more expressive representation of the basis functions, leading to a more accurate solution approximation.

To address the issue with the boundary conditions, we introduced a final version of the architecture, with the addition of sine augmentation blocks (Figure 6.16). This involved multiplying the DeepONet's outputs with functions that vanish at the boundaries, while remaining close to one in the interior of the domain. This ensured the model adheres to the boundary conditions: $u_1, u_2 = 0$ at the top and right boundaries, $u_1 = 0$ at the left boundary, and $u_2 = 0$ at the bottom boundary. Evaluation of the trained DeepONet on the test set revealed an R^2 score of 0.996, an aRRMSE of 0.0633, and an aRelErr of 0.0315. This marked a small overall performance increase compared to the previous version and proved to be the best-performing architecture we considered for multiple wound shapes. Importantly, the sine augmentation effectively resolved the boundary issues, without compromising the model's predictions in the domain. The RSAW curves in Figure 6.19 confirmed that prediction accuracy at the wound boundary was not diminished by the sine augmentation. The worst-case prediction remained consistent with previous results, while the best-case prediction showed slight improvement, with a 1.7% underestimation of the minimum RSAW and a 2% overestimation of the final value at $t = 100$.

We conclude that DeepONets, with tailored architecture adjustments, can serve as accurate surrogates for finite element solutions when applied to basic initial wound geometries with variable sizes.

8.3. Predictions on Convex Combinations of Wound Shapes

The final step in answering our main research question was to investigate how well the final DeepONet generalises to convex combinations of the three basic geometries. To find out, we generated a new test set including finite element simulations based on convex combinations of the three basic wound shapes. Upon evaluation on this convex test set, we found that the predictions closely matched targets, demonstrating good generalisation. The scatter plots in Figure 7.3 confirm this, showing a strong concentration around $y = x$ with minimal outliers, very similar to the results on the original test set. The model slightly underperformed on mid-range y-displacements, but excelled in predicting both small and large displacements. The RSAW curves in Figure 7.4 also showed that the model's performance on the convex test set is comparable to the original test set, with some improvements in the best case. The error analysis in Figure 7.4c showed a gradual increase in absolute error over time, peaking around day 90. However, the standard deviation of the error never exceeds 3%, which corresponds to deviations on the order of tenths of millimeters. From a practical medical perspective, these are very accurate

predictions.

We compared our various DeepONet architectures on the convex test set (Figure 7.5). The final model, incorporating initial wound shape info and sine augmentation, achieved the best performance with an R^2 score of 0.9944, an aRRMSE of 0.075, and an aRelErr of 0.0343. While performance on the convex test set was slightly lower than on the original test set, the differences were minimal. This confirms the model's strong generalisation abilities. The results support that initial shape info should be input to the trunk, and sine augmentation enhances performance when the model is already well-tuned.

Lastly, we investigated extending DeepONet predictions to one year, rather than just 100 days. We expanded the training set with 50 new finite element simulations, where $t_{end} = 365$. Initialising the network with the learned parameters from the model trained on $t \in [0, 100]$ proved to be effective. We compared the RSAW curves for two scenarios: one where 50 finite element simulations were added to the training set, with 30 time steps sampled in $t \in (100, 365]$, and another where 150 runs were added, with 10 time steps each. We found that adding new, unique data (the first scenario) results in the greatest improvement. The is likely because introducing new information provides a more varied learning experience, whereas additional data points from existing samples tend to add redundancy, offering less benefit.

We conclude that we have successfully trained a DeepONet as a finite element surrogate, capable of accurately predicting the dermal displacement field. The training encompassed multiple initial wound shapes, with the network demonstrating strong generalisation on convex combinations of these basic shapes. Furthermore, by adding a limited number of samples from one-year simulations into the training set, the network achieved reasonable predictions of the RSAW over the entire year. We anticipate that including more data with $t_{end} = 365$ will further enhance prediction accuracy.

8.4. Conclusion

At the onset of this study, we formulated the following main research question: can we train a neural operator based on the DeepONet architecture to accurately predict post-burn wound evolution over time, while accounting for multiple initial wound shapes? We conclude that DeepONets can indeed serve as effective surrogates for the finite element simulations, demonstrating both accuracy and efficiency in predicting dermal displacement. When trained on a single wound shape, the model demonstrated strong performance. Expanding to multiple wound shapes, DeepONets with tailored architectures, including initial wound shape information and sine augmentation, showed significant improvement in accuracy. Finally, we found that the model successfully generalised to convex combinations of basic shapes and provided reasonable predictions over an extended period of one year.

9 Discussion

This chapter addresses the limitations of our work and formulates directions for further research. Sections 9.1 to 9.6 consider various aspects, including the limitations in DeepONet architecture, it's inputs and outputs, the considered wound shapes, training and hyperparameter tuning, one-year predictions, and the morphoelastic model.

9.1. DeepONet Architecture

In this research, we explored and compared several DeepONet architectures, although we maintained a fixed basic configuration. We believe there is considerable room for exploring alternative designs. One area for further research is the comparison between different strategies for splitting the branch and trunk networks. Although we briefly considered splitting either the branch or trunk network while sharing the other, more comprehensive investigations could include training two independent DeepONets, or splitting both branch and trunk. Additionally, a comparative study with stacked versus unstacked DeepONets could provide insights into whether our choice of using the latter was optimal. Moreover, the specific architecture of the branch and trunk networks are areas that could benefit from further investigation. Our initial experiments only varied the number of output nodes in both networks, leaving the width and depth of both networks unexplored.

To further validate the effectiveness of DeepONets, comparisons with other neural operators, such as the Fourier neural operator, are a direction for further research. Such comparisons might reveal if alternative approaches can give better performance or if different architectures could provide additional advantages. Additionally, integrating DeepONets with PINNs can ensure that the predicted displacements adhere more closely to the governing PDEs, which now is not necessarily the case. This could solve potential discrepancies between the predictions and the underlying physical laws, leading to more reliable outcomes.

9.2. Inputs and Outputs

The aim of a DeepONet is to learn the solution operator to a (system of) PDE(s), effectively mapping the parameter space to the solution space. In our context, this means that given any set of the five parameters from the morphoelastic model, the DeepONet predicts the displacements at any point (t, x, y). However, our study primarily focused on the efficacy of the DeepONets across different wound shapes, without much emphasis on the five input parameters. A more detailed and in-depth investigation into how these parameters influence predictions was not conducted. We uniformly sample them from prespecified ranges (Table 5.1), based on a stability analysis and sensitivity study by Egberts et al. [19, 21]. However, these parameters are likely correlated, and the uniform sampling may lead to unrealistic predictions. It would be valuable to investigate more representative methods of drawing the input values within the ranges. For example, sampling parameters from a function space rather than treating them as constants might have improved model applicability and accuracy for diverse patient scenarios. Additionally, extending the parameter set was something we could have explored. We chose to input

five patient-specific parameters to the branch network, without much consideration of their physical significance. Including a broader range of patient-specific parameters, perhaps all of them, could have provided a more comprehensive and applicable model. This approach would likely result in larger variability in the solutions, presenting an interesting opportunity to test whether the model can manage this increased complexity.

At the beginning of our research, we decided to focus on predicting the dermal displacement field over time, which resulted in a DeepONet with two outputs. An alternative way of increasing the model's complexity is to have the DeepONet predict more than just the displacement field. The finite element simulations solve for four biological constituents and three mechanical components, offering ample opportunity for additional outputs. For instance, the strain energy, computed as a post-processing step from the effective strain and collagen concentration, could be included as an extra output. Since strain energy is a measure of a patient's discomfort [18], adding it to the prediction could enhance our model's relevance. However, adding more outputs would introduce a new set of challenges regarding the design of the architecture.

9.3. Wound Shapes

A key aspect of our research was the novelty of incorporating multiple initial wound shapes. We successfully managed this for three basic shapes and found that the model generalises well to convex combinations of these shapes. A promising direction for further research is to investigate the application of more complex and realistic initial wound geometries, where for example no symmetry in x and y is present. We did not test the DeepONet's performance when subjected to a more random initial wound shape, outside the convex hull of the three basic shapes. As the network has never seen something like this during training, we anticipate a significant decrease in performance under these conditions. To address this, we could think of alternative ways of incorporating the wound geometry into the DeepONet. For example, we might parameterise the wound shape and use this as input to the branch network, allowing the DeepONet to learn the mapping from the wound geometry to the solution space. Of course, for this to be effective, the numerical model must also be capable of handling such complex wound shapes, as we would still need to compare our predictions against a reliable baseline. This is not yet the case.

9.4. Training and Hyperparameter Optimisation

A more systematic hyperparameter optimisation approach could have potentially improved our model's performance. Our approach to it was somewhat preliminary and we did not perform very extensive tests. For instance, we did not compare different initialisation schemes or activation functions. The hyperparameter tuning we performed was focused on the initial model that simulated the displacement field for one wound shape, and since we were satisfied with its performance, we refrained from tuning further, even as we frequently altered the architecture. It could have been advantageous to perform a rigorous hyperparameter search for the final model. Furthermore, there a several techniques that we could have explored, including *cross-validation*, *adaptive learning rates*, and *regularisation* [29]. Cross-validation might have offered a more robust evaluation of the model's generalisability. Adaptive learning rates could improve training efficiency, while regularisation techniques might enhance model robustness by mitigating overfitting. We leave this as recommendations for future research.

9.5. One-Year Predictions

We initially chose to limit our predictions to 100 days, rather than an entire year, as this was more time efficient for generating large datasets. This constraint resulted in some loss of information, particularly regarding the asymptotic value of the RSAW, which is not always captured within 100 days. A more comprehensive approach would have been running all simulations for an entire year, to obtain a complete picture of the RSAW evolution. In our final test, we did include data with $t_{end} = 365$ and demonstrated that the RSAW could be predicted reasonably well over an entire year. We found that adding new finite element samples was more beneficial than drawing more data points per sample. However, a more extensive evaluation, using larger additional training data and more fine-grained sampling, could have provided a clearer picture of the optimal approach. Performing a thorough comparison of performance metrics for the different strategies would also have offered a clearer assessment of the best approach.

This is recommendation for future research.

9.6. Morphoelastic Model

In this study, we compared our predictions against finite element simulations of the morphoelastic model, assuming these simulations reflect the truth. However, we did not account for the inherent numerical errors. Improving the numerical model could lead to more detailed and accurate predictions. The two-dimensional morphoelastic model we used in this study is still under development and might benefit from further refinement. As noted by Egberts et al. [18], incorporating the distinct collagen types and integrating the immune system's role in the wound healing process could be beneficial, as these are factors influencing post-burn healing and scar formation. Additionally, exploring three-dimensional models could give a more detailed simulation by accounting for the wound depth. However, this requires balancing accuracy with computational efficiency, as a 3D model would significantly increase the computational complexity. To manage this, we may need to consider techniques such as using rotational symmetry, making it feasible to generate large training datasets.

For generating our current training, evaluation, and test sets, we relied on the numerical model as well. Ideally, we would prefer to use real patient-specific data for this purpose. However, collecting such clinical datasets from anonymous patients presents significant challenges and would involve the cooperation from multiple parties. According to Egberts et al., this is still a work in progress.

References

[1] B. Alberts et al. *The molecular biology of the cell*. Second. Garland Publishing, 1989.

[2] Filippo Amato et al. *Artificial neural networks in medical diagnosis*. 2013.

[3] George Keith Batchelor. *An introduction to fluid dynamics*. Cambridge university press, 2000.

[4] WM Boon, DC Koppenol, and FJ Vermolen. "A multi-agent cell-based model for wound contraction". In: *Journal of biomechanics* 49.8 (2016), pp. 1388–1401.

[5] Ralph A Bradshaw and Edward A Dennis. *Handbook of cell signaling*. Academic press, 2009.

[6] Shengze Cai et al. "DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks". In: *Journal of Computational Physics* 436 (2021), p. 110296.

[7] Anthony L Caterini and Dong Eui Chang. *Deep neural networks in a mathematical framework*. Springer, 2018.

[8] Tianping Chen and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems". In: *IEEE transactions on neural networks* 6.4 (1995), pp. 911–917.

[9] Gloria A Chin, Robert F Diegelmann, and Gregory S Schultz. "Cellular and molecular regulation of wound healing". In: *Basic and Clinical Dermatology* 33 (2005), p. 17.

[10] A. Desmoulière et al. "Transforming growth factor-β1 induces alpha-smooth muscle actin expression in granulation tissue myofibroblasts and in quiescent and growing cultured fibroblasts". In: *The Journal of Cell Biology* 122.1 (1993), pp. 103–111. DOI: 10.1083/jcb.122.1.103.

[11] Milan Despotovic et al. "Evaluation of empirical models for predicting monthly mean horizontal diffuse solar radiation". In: *Renewable and Sustainable Energy Reviews* 56 (2016), pp. 246–260.

[12] Patricio Clark Di Leoni et al. "Neural operator prediction of linear instability waves in high-speed boundary layers". In: *Journal of Computational Physics* 474 (2023), p. 111793.

[13] Waleed Diab and Mohammed Al-Kobaisi. "U-DeepONet: U-Net Enhanced Deep Operator Network for Geologic Carbon Sequestration". In: *arXiv preprint arXiv:2311.15288* (2023).

[14] Robert F Diegelmann, Melissa C Evans, et al. "Wound healing: an overview of acute, fibrotic and delayed healing". In: *Front biosci* 9.1 (2004), pp. 283–289.

[15] Timothy Dozat. "Incorporating nesterov momentum into adam". In: (2016). URL: https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ.

[16] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[17] Ginger Egberts, Fred Vermolen, and Paul Van Zuijlen. "A one-dimensional morphoelastic model for burn injuries: stability analysis, numerical validation and biological interpretation". In: *arXiv preprint arXiv:2010.12897* (2020).

[18] Ginger Egberts, Fred Vermolen, and Paul van Zuijlen. "High-speed predictions of post-burn contraction using a neural network trained on 2D-finite element simulations". In: *Frontiers in Applied Mathematics and Statistics* 9 (2023), p. 1098242.

[19] Ginger Egberts, Fred Vermolen, and Paul van Zuijlen. "Stability of a two-dimensional biomorphoelastic model for post-burn contraction". In: *Journal of Mathematical Biology* 86.4 (2023), p. 59.

[20] Ginger Egberts et al. "A Bayesian finite-element trained machine learning approach for predicting post-burn contraction". In: *Neural Computing and Applications* 34.11 (2022), pp. 8635–8642.

[21] Ginger Egberts et al. “Sensitivity of a two-dimensional biomorphoelastic model for post-burn contraction”. In: *Biomechanics and Modeling in Mechanobiology* 22.1 (2023), pp. 105–121.

[22] Madeleine Flanagan. *Wound healing and skin integrity: principles and practice*. John Wiley & Sons, 2013.

[23] Ruth K Freinkel and David T Woodley. *The biology of the skin*. CRC Press, 2001.

[24] David Gawkrodger. *Dermatology e-book: an illustrated colour text*. Elsevier Health Sciences, 2016.

[25] II George Broughton, Jeffrey E Janis, and Christopher E Attinger. “The basic science of wound healing”. In: *Plastic and reconstructive surgery* 117.7S (2006), 12S–34S.

[26] Roxana Ghiulai et al. “Tetracyclic and pentacyclic triterpenes with high therapeutic efficiency in wound healing approaches”. In: *Molecules* 25.23 (2020), p. 5557.

[27] GJ Glas, M Levi, and MJ Schultz. “Coagulopathy and its management in patients with severe burns”. In: *Journal of Thrombosis and Haemostasis* 14.5 (2016), pp. 865–874.

[28] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[30] K. Gosh et al. “Cell adaptation to a physiologically relevant ECM mimic with different viscoelastic properties”. In: *Biomaterials* 28.4 (2007), pp. 671–679. DOI: 10.1016/j.biomaterials.2006.09.038.

[31] G. Grotendorst. “Chemoattractants and growth factors”. In: 1st ed. W.B. Saunders, Philidelphia, Pennsylvania, 1992. Chap. 15, pp. 237–246.

[32] Cameron Luke Hall. “Modelling of some biological materials using continuum mechanics”. PhD thesis. Queensland University of Technology, 2008.

[33] J. Haugh. “Deterministic model of dermal wound invasion incorporating receptor-mediated signal transduction and spatial gradient sensing”. In: *Biophysical Journal* 90.7 (2006), pp. 2297–2308. DOI: 10.1529/biophysj.105.077610.

[34] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[35] Alexander Heinlein and Krzysztof Postek. *Linear Algebra and Optimization for Machine Learning*. TU Delft, June 2022.

[36] Thomas Hillen and Kevin J Painter. “A user’s guide to PDE models for chemotaxis”. In: *Journal of mathematical biology* 58.1-2 (2009), pp. 183–217.

[37] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent”. In: *Cited on* 14.8 (2012), p. 2.

[38] Boris Hinz. “Formation and function of the myofibroblast during tissue repair”. In: *Journal of Investigative Dermatology* 127.3 (2007), pp. 526–537.

[39] Elisabeth Hofmann et al. “Modelling the complexity of human skin in vitro”. In: *Biomedicines* 11.3 (2023), p. 794.

[40] Icrp. *ICRP Publication 110*. Table A.1 p. 51. SAGE Publications Ltd, 2010. 168 pp. ISBN: 0702041866.

[41] E Javierre et al. “Numerical modeling of a mechano-chemical theory for wound contraction analysis”. In: *International journal of solids and structures* 46.20 (2009), pp. 3597–3606.

[42] Jianmin Jiang, P Trundle, and Jinchang Ren. “Medical image analysis with artificial neural networks”. In: *Computerized Medical Imaging and Graphics* 34.8 (2010), pp. 617–631.

[43] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[44] D. Koppenol and F. Vermolen. “Biomedical implications from a morphoelastic continuum model for the simulation of contracture formation in skin grafts that cover excised burns”. In: *Biomechanics and Modeling in Mechanobiology* 16.4 (2017), pp. 1187–1206. DOI: 10.1007/s10237-017-0881-y.

[45] Daniël C Koppenol et al. “A mathematical model for the simulation of the formation and the subsequent regression of hypertrophic scar tissue after dermal wounding”. In: *Biomechanics and modeling in mechanobiology* 16 (2017), pp. 15–32.

[46] DC Koppenol. “Biomedical Implications from Mathematical Models for the Simulation of Dermal Wound Healing”. PhD thesis. Delft, Netherlands: Delft University of Technology, 2017. URL: https://repository.tudelft.nl/record/uuid:cdc7392f-4ac6-404c-9615-dc425f67efae.

[47] Soheila S. Kordestani. “Chapter 3 - Wound Healing Process”. In: *Atlas of Wound Healing*. Ed. by Soheila S. Kordestani. Elsevier, 2019, pp. 11–22.

[48] Nikola Kovachki et al. “Neural operator: Learning maps between function spaces with applications to pdes”. In: *Journal of Machine Learning Research* 24.89 (2023), pp. 1–97.

[49] Gaoyang Li et al. “Prediction of 3D Cardiovascular hemodynamics before and after coronary artery bypass surgery via deep learning”. In: *Communications biology* 4.1 (2021), p. 99.

[50] Zongyi Li et al. “Fourier neural operator for parametric partial differential equations”. In: *arXiv preprint arXiv:2010.08895* (2020).

[51] Liang Liang et al. “A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis”. In: *Journal of The Royal Society Interface* 15.138 (2018), p. 20170844.

[52] Liang Liang et al. “A machine learning approach as a surrogate of finite element analysis–based inverse method to estimate the zero-pressure geometry of human thoracic aorta”. In: *International journal for numerical methods in biomedical engineering* 34.8 (2018), e3103.

[53] Chensen Lin et al. “Operator learning for predicting multiscale bubble growth dynamics”. In: *The Journal of Chemical Physics* 154.10 (2021).

[54] Lu Lu, Pengzhan Jin, and George Em Karniadakis. “Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators”. In: *arXiv preprint arXiv:1910.03193* (2019).

[55] Pengfei Lu et al. “Extracellular matrix degradation and remodeling in development and disease”. In: *Cold Spring Harbor perspectives in biology* 3.12 (2011), a005058.

[56] Ali Madani et al. “Bridging finite element and machine learning modeling: stress prediction of arterial walls in atherosclerosis”. In: *Journal of biomechanical engineering* 141.8 (2019), p. 084502.

[57] Guido Majno and Isabelle Joris. *Cells, tissues, and disease: principles of general pathology*. Oxford University Press, 2004.

[58] Zhiping Mao et al. “DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators”. In: *Journal of computational physics* 447 (2021), p. 110698.

[59] S. Maskarinec et al. “Quantifying cellular traction forces in three dimensions”. In: *Proceedings of the National Academy of Sciences* 106.52 (2009), pp. 22108–22113. DOI: 10.1073/pnas.0904565106.

[60] JA McGrath, RAJ Eady, and FM Pope. “Anatomy and organization of human skin”. In: *Rook’s textbook of dermatology* 1 (2004), pp. 3–2.

[61] William Montagna. *The structure and function of skin*. Elsevier, 2012.

[62] Ken’ichi Morooka et al. “Real-time nonlinear FEM with neural network for simulating soft organ model deformation”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2008: 11th International Conference, New York, NY, USA, September 6-10, 2008, Proceedings, Part II 11*. Springer. 2008, pp. 742–749.

[63] Christian Moya et al. “Deeponet-grid-uq: A trustworthy deep operator framework for predicting the power grid’s post-fault trajectories”. In: *Neurocomputing* 535 (2023), pp. 166–182.

[64] K. Murphy et al. “A fibrocontractive mechanochemical model of dermal wound closure incorporating realistic growth factor kinetics”. In: *Bulletin of Mathematical Biology* 74.5 (2012), pp. 1143–1170. DOI: 10.1007/s11538-011-9712-y.

[65] Ali Narin, Ceren Kaya, and Ziynet Pamuk. “Automatic detection of coronavirus disease (covid-19) using x-ray images and deep convolutional neural networks”. In: *Pattern Analysis and Applications* 24 (2021), pp. 1207–1220.

[66] Nederlandse Brandwonden Stichting. *Over Brandwonden*. https://brandwondenstichting.nl/over-brandwonden/. Accessed: 2024-08-18. 2024.

[67] Denis Noble. “The rise of computational biology”. In: *Nature Reviews Molecular Cell Biology* 3.6 (2002), pp. 459–463.

[68] L. Olsen, J. Sherratt, and P. Maini. “A mechanochemical model for adult dermal wound contraction and the permanence of the contracted tissue displacement profile”. In: *Journal of Theoretical Biology* 177.2 (1995), pp. 113–128. DOI: 10.1006/jtbi.1995.0230.

[69] Julian D Osorio et al. “Forecasting solar-thermal systems performance under transient operation using a data-driven machine learning approach based on the deep operator network architecture”. In: *Energy Conversion and Management* 252 (2022), p. 115063.

[70] C. Overall, J. Wrana, and J. Sodek. “Transcriptional and post-transcriptional regulation of 72-kda gelatinase/ type IV collagenase by transforming growth factor-beta in human fibroblasts”. In: *Journal of Biological Chemistry* 266.21 (1991), pp. 14061–14071.

[71] Oscar J Pellicer-Valero et al. “Real-time biomechanical modeling of the liver using machine learning models trained on finite element method simulations”. In: *Expert Systems with Applications* 143 (2020), p. 113083.

[72] Rebecca Penzer and Steven Ersser. *Principles of skin care: a guide for nurses and health care practitioners*. John Wiley & Sons, 2010.

[73] Micha Pfeiffer et al. “Learning soft tissue behavior of organs for surgical navigation with convolutional neural networks”. In: *International journal of computer assisted radiology and surgery* 14 (2019), pp. 1147–1155.

[74] Karin Pfisterer et al. “The extracellular matrix in skin inflammation and infection”. In: *Frontiers in cell and developmental biology* 9 (2021), p. 682414.

[75] W John Pugh and Robert P Chilcott. “Principles of diffusion and thermodynamics”. In: *Principles and practice of skin toxicology* (2008), pp. 93–107.

[76] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[77] Bogdan Raonic et al. “Convolutional neural operators for robust and accurate learning of PDEs”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[78] Erfan Rezvani Ghomi et al. “Wound dressings: Current advances and future directions”. In: *Journal of Applied Polymer Science* 136.27 (2019), p. 47738.

[79] A. Roberts et al. “Transforming growth factor type beta: rapid induction of fibrosis and angiogenesis in vivo and stimulation of collagen formation in vitro.” In: *Proceedings of the National Academy of Sciences* 83.12 (1986), pp. 4167–4171. DOI: 10.1073/pnas.83.12.4167.

[80] Kathleen S Romanowski et al. “American Burn Association guidelines on the management of acute pain in the adult burn patient: a review of the literature, a compilation of expert opinion, and next steps”. In: *Journal of Burn Care & Research* 41.6 (2020), pp. 1129–1151.

[81] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[82] R. Rudolph and J. Vande Berg. “The myofibroblast in Dupuytren’s contracture”. In: *Journal of Hand Clinics* 7.4 (1991). Discussion 693-4, pp. 683–692.

[83] Marianne Schaaphok. “A Fast Neural Network-Based Computational Framework for the Prediction of Skin Contraction”. Unpublished thesis. 2020.

[84] Tom Schaul, Ioannis Antonoglou, and David Silver. “Unit tests for stochastic optimization”. In: *arXiv preprint arXiv:1312.6055* (2013).

[85] P Mohamed Shakeel et al. “Neural network based brain tumor detection using wireless infrared imaging sensor”. In: *IEEE Access* 7 (2019), pp. 5577–5588.

[86] Olga Sierawska et al. “Innate immune system response to burn damage—focus on cytokine alteration”. In: *International journal of molecular sciences* 23.2 (2022), p. 716.

[87] A. Sillman et al. “Human dermal fibroblasts do not exhibit directional migration on collagen I in direct-current electric fields of physiological strength”. In: *Experimental Dermatology* 12.4 (2003), pp. 396–402. DOI: 10.1034/j.1600-0625.2002.120406.x.

[88] F. Strutz et al. “TGF-β1 induces proliferation in human renal fibroblasts via induction of basic fibroblast growth factor (FGF-2)”. In: *Kidney International* 59.2 (2001), pp. 579–592. DOI: 10.1046/j.1523-1755.2001.059002579.x.

[89] Yixuan Sun et al. “Deepgraphonet: A deep graph operator network to learn and zero-shot transfer the dynamic response of networked systems”. In: *IEEE Systems Journal* (2023).

[90] Agnieszka Surowiecka, Tomasz Korzeniowski, and Jerzy Strużyna. “Early burn wound excision in mass casualty events”. In: *Military Medical Research* 9.1 (2022), p. 42.

[91] John Lighton Synge and Alfred Schild. *Tensor calculus*. Vol. 5. Courier Corporation, 1978.

[92] J. Vande Berg et al. “Comparative growth dynamics and actin concentration between cultured human myofibroblasts from granulating wounds and dermal fibroblasts from normal skin”. In: *Lab Invest* 61.5 (1989), pp. 532–538.

[93] F.J. Vermolen. “Chapter 1 - Mathematical models of healing of burns”. In: *Innovations and Emerging Technologies in Wound Care*. Ed. by Amit Gefen. Academic Press, 2020, pp. 1–20. ISBN: 978-0-12-815028-3.

[94] FJ Vermolen and A Gefen. “Semi-stochastic cell-level computational modelling of cellular forces: application to contractures in burns and cyclic loading”. In: *Biomechanics and modeling in mechanobiology* 14 (2015), pp. 1181–1195.

[95] Fred Vermolen and Ilkka Pölönen. “Uncertainty quantification on a spatial Markov-chain model for the progression of skin cancer”. In: *Journal of mathematical biology* 80.3 (2020), pp. 545–573.

[96] Komal Vig et al. “Advances in skin regeneration using tissue engineering”. In: *International journal of molecular sciences* 18.4 (2017), p. 789.

[97] Sifan Wang, Hanwen Wang, and Paris Perdikaris. “Learning the solution operator of parametric partial differential equations with physics-informed DeepONets”. In: *Science advances* 7.40 (2021), eabi8605.

[98] World Health Organization. *Burns*. https://www.who.int/news-room/fact-sheets/detail/burns. Accessed: 2024-08-18. 2024.

[99] L. Wrobel et al. “Contractility of single human dermal myofibroblasts and fibroblasts”. In: *Cell Motility and the Cytoskeleton* 52.2 (2002), pp. 82–90. DOI: 10.1002/cm.10034.

[100] Minglang Yin et al. “Simulating progressive intramural damage leading to aortic dissection using DeepONet: an operator–regression neural network”. In: *Journal of the Royal Society Interface* 19.187 (2022), p. 20210670.

A

Parameter Values Morphoelastic Model

Table A.1: Overview of the fixed parameter values used for the numerical simulations. 'NC' indicates that the parameter value is a consequence of the chosen values for other parameters

Symbol	Value	Dimension	Reference
D_F	10^{-7}	cm^5/(cells day)	[87]
D_c	2.9×10^{-3}	cm^2/day	[33]
χ_F	2×10^{-3}	cm^5/(g day)	[64]
k_c	4×10^{-13}	g/(cells day)	[68]
r_F	9.24×10^{-1}	cm^{3q}/(cells^q day)	[1], [30]
$r_F^{\max}$	2	-	[88]
k_ρ	7.6×10^{-8}	g/(cells day)	[NC]
$k_\rho^{\max}$	10	-	[68]
a_c^I	10^{-8}	g/cm^3	[31], [68]
a_c^{II}	10^{-8}	g/cm^3	[68]
a_c^{III}	2×10^8	cm^3/g	[70]
a_c^{IV}	10^{-9}	g/cm^3	[79]
η^I	2	-	[82]
η^{II}	5×10^{-1}	-	[44]
k_F	1.08×10^7	cm^3/(g day)	[10]
κ_F	10^{-6}	cm^3/cells	[92]
q	-4.151×10^{-1}	-	[NC]
δ_N	2×10^{-2}	/day	[68]
δ_M	6×10^{-2}	/day	[45]
δ_c	5×10^{-4}	cm^6/(cells g day)	[68]
δ_ρ	6×10^{-6}	cm^6/(cells g day)	[45]
$\overline{N}$	10^4	cells/cm^3	[68]
$\overline{M}$	0	cells/cm^3	[68]
$\overline{c}$	0	g/cm^3	[45]
$\overline{\rho}$	1.125×10^{-1}	g/cm^3	[68]
ρ_t	1.09	g/cm^3	[40]
μ_1	10^2	(N day)/cm^2	[44]
μ_2	10^2	(N day)/cm^2	[44]
E	32	N/((g cm)$^{0.5}$)	[21]
ξ	5×10^{-2}	(N g)/(cells cm^2)	[59], [99]
R	9.95×10^{-1}	g/cm^3	[44]

Continued on next page

Table A.1: (continued)

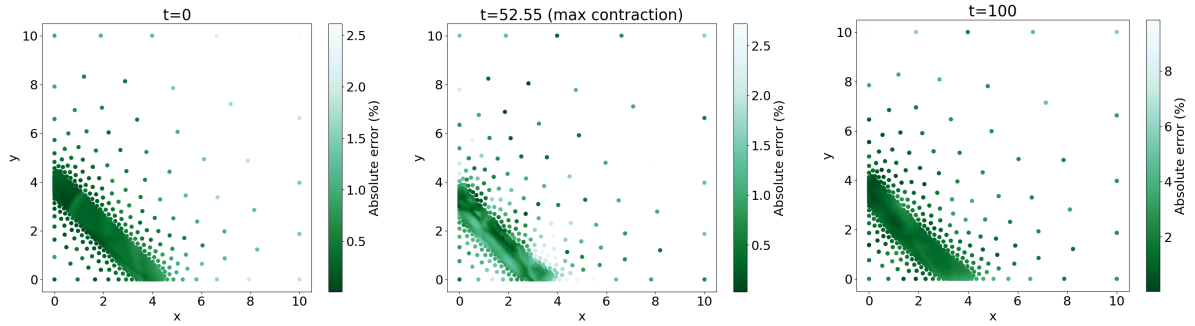
Symbol	Value	Dimension	Reference
ζ	4×10^2	cm^6/(cells g day)	[44]
ν	4.9×10^{-1}	-	[21]
$\widetilde{N}$	2×10^3	cells/cm^3	[NC]
$\widetilde{c}$	10^{-8}	g/cm^3	[21]
$\widetilde{\rho}$	1.13×10^{-2}	g/cm^3	[NC]

B

Absolute Error Plots

This appendix presents the absolute error plots for the different DeepONet architectures, analysing a single wound shape, multiple wound shapes, and convex combinations of wound shapes.

Single Wound Shape

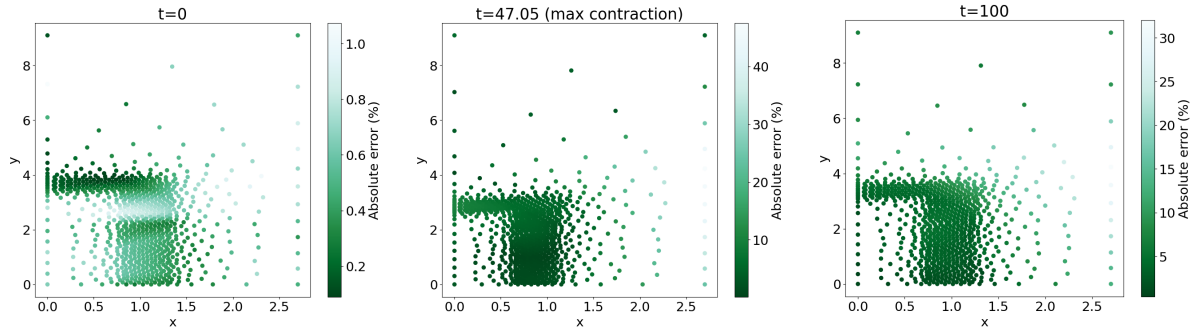


(a) **(b)** **(c)**

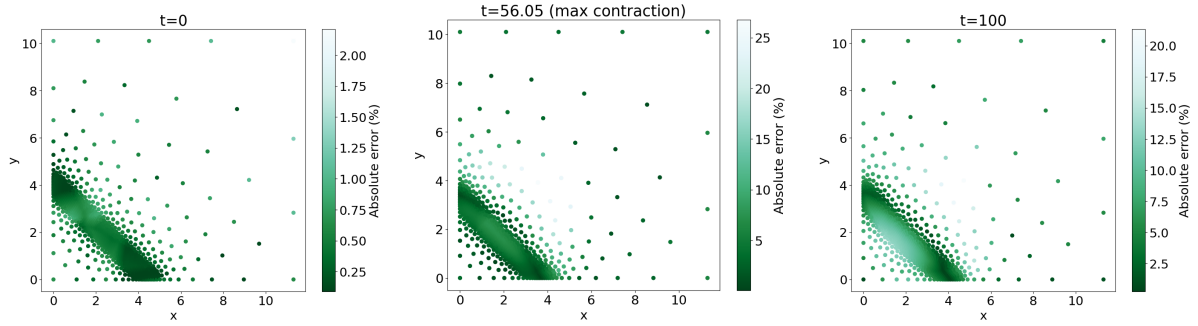
Figure B.1: Average absolute error at $t = 0$, at the time of maximal contraction, and at $t = 100$. Corresponds to sample in Figure 5.8.

Multiple Wound Shapes

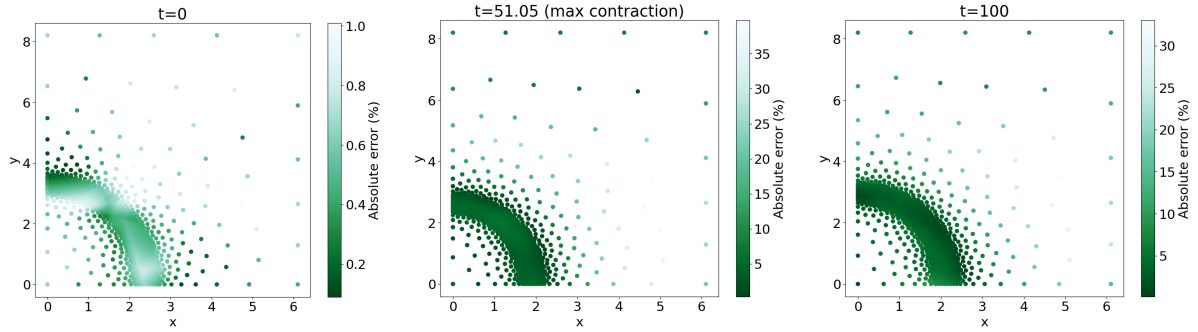
Skeleton DeepONet



(a) Rectangular-shaped wound



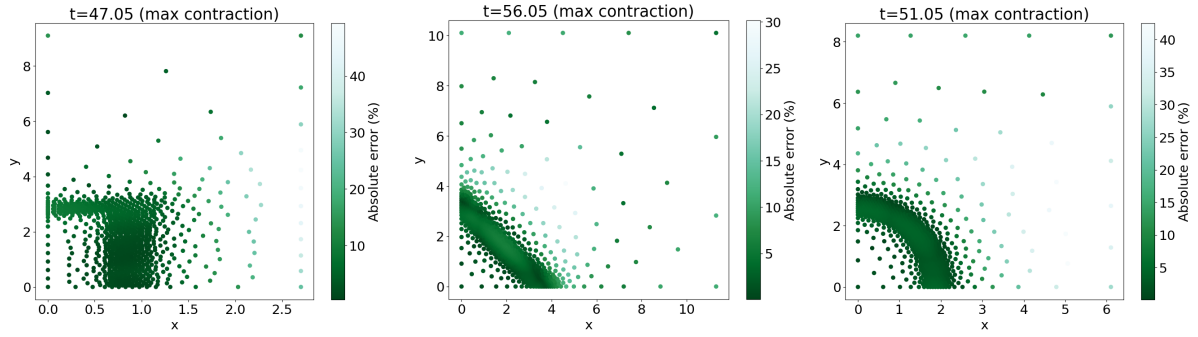
(a) Rhombus-shaped wound



(b) Elliptic-shaped wound

Figure B.3: Average absolute error for three samples from the test set at $t = 0$, at the time of maximal contraction, and at $t = 100$. Corresponds to Figure 6.3.

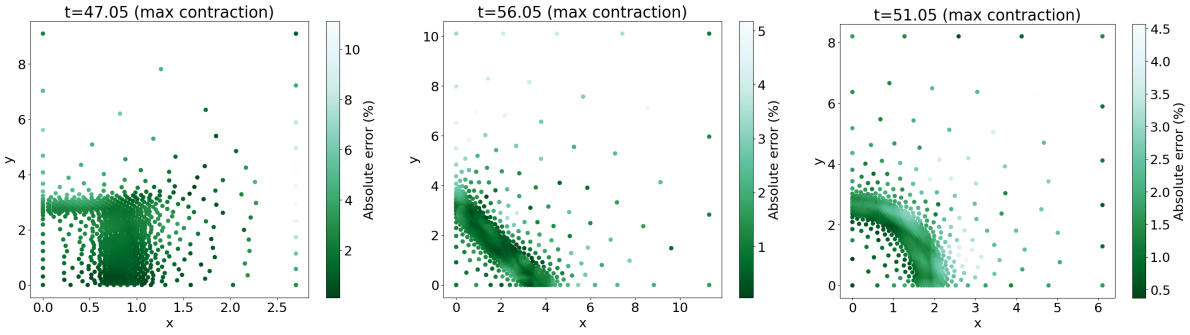
Addition to the Branch Network



(a) Rectangular-shaped wound **(b)** Rhombus-shaped wound **(c)** Elliptic-shaped wound

Figure B.4: Average absolute error for three samples from the test set, at their respective time of maximal contraction. Corresponds to Figure 6.7.

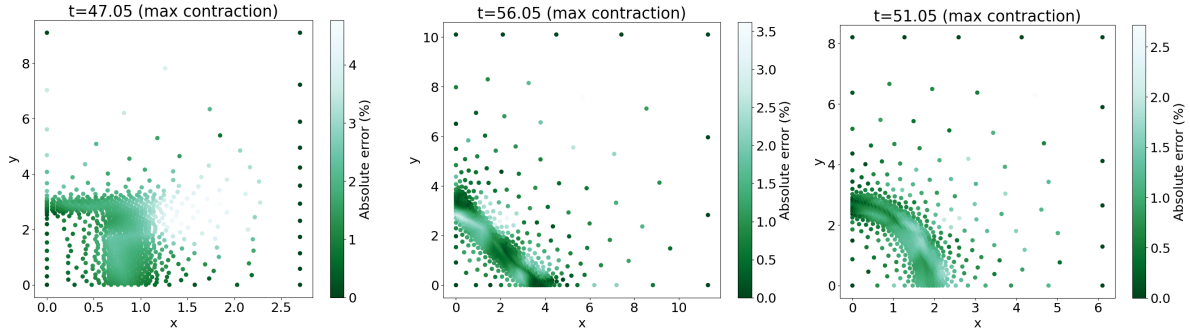
Addition to the Trunk Network



(a) Rectangular-shaped wound **(b)** Rhombus-shaped wound **(c)** Elliptic-shaped wound

Figure B.5: Average absolute error for three samples from the test set, at their respective time of maximal contraction. Corresponds to Figure 6.12.

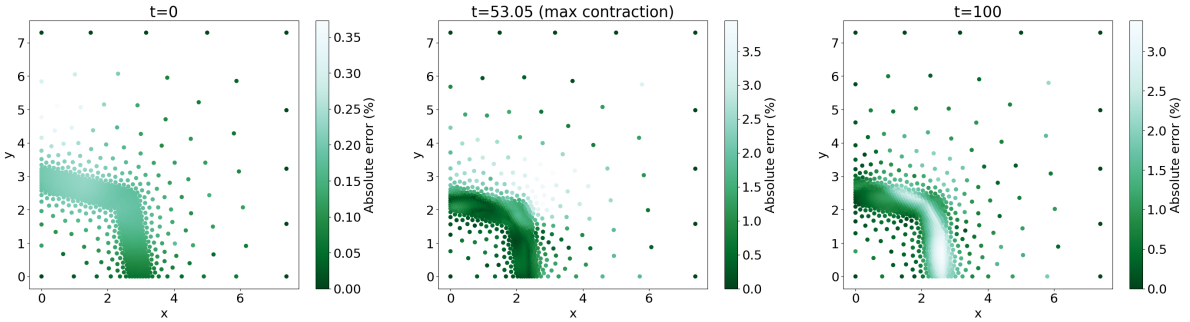
Sine Augmentation



(a) Rectangular-shaped wound **(b)** Rhombus-shaped wound **(c)** Elliptic-shaped wound

Figure B.6: Average absolute error for three samples from the test set, at their respective time of maximal contraction. Corresponds to Figure 6.17.

Convex Combinations of Wound Shapes



$0.44 \cdot \text{rectangle} + 0.49 \cdot \text{rhombus} + 0.07 \cdot \text{ellipse}$

Figure B.7: Average absolute error at $t = 0$, at the time of maximal contraction, and at $t = 100$. Corresponds to sample in Figure 7.2.