Deep Learning of Inverse Dynamic Models

Deep Learning von Inversen Dynamik Modellen
Master-Thesis von Christian Ritter aus Aschaffenburg
Oktober 2018

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Deep Learning of Inverse Dynamic Models
Deep Learning von Inversen Dynamik Modellen

Vorgelegte Master-Thesis von Christian Ritter aus Aschaffenburg

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

Tag der Einreichung:

For Lisa

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 30. Oktober 2018

(Christian Ritter)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, October 30, 2018

(Christian Ritter)

Abstract

In the last few years, deep learning has alleviated most fields of machine learning to new levels. However, deep learning has yet to prove its value in the field of model learning in robotics. Currently, dynamics models have to be derived analytically in a tedious process of measuring the mechanical system part by part. In applications like automated industrial production where high precision and reliability are required, robot movements are engineered precisely costing money and time. The reason why the models are not learned may be the fact that data-based approaches lack the physical plausibility of analytical models. In this thesis I propose Deep Lagrangian Network (DeLaN) a deep learning approach to inverse dynamics models. The novelty of DeLaN is the incorporation of a physics prior, the Lagrange-Euler partial differential equation. Thus, DeLaN encodes physical knowledge while facilitating standard model learning advantages, like end-to-end training using data samples. I compare my proposed approach to a standard feed-forward neural network (FF-NN) in various experiments to showcase the usefulness of DeLaN in learning inverse dynamics models online. I am also highlighting advantages and limitations of my approach based on the characteristics inherited from the physics prior. In a nutshell, DeLaN is able to learn and control a real robot and provides capabilities to extrapolate to unseen tasks.

Zusammenfassung

In den letzten Jahren hat Deep Learning viele Bereiche des Machine Learnings auf neue Ebenen gehoben. Deep Learning muss seinen Wert für Model Learning in der Robotik allerdings noch beweisen. Dynamische Modelle werden aktuell noch in einem aufwendigen Prozess analytisch hergeleitet, wobei das mechanische System Stück für Stück ausgemessen werden muss. In Anwendungen, wie der automatisierten industriellen Produktion, die hohe Präzision und Zuverlässigkeit benötigen, werden Roboterbewegungen präzise von Ingenieuren modelliert, was Zeit und Geld kostet. Der Grund, warum die Modelle aktuell nicht gelernt werden, könnte der Fakt sein, dass datenbasierte Ansätze nicht die physikalische Plausibilität analytischer Modelle bieten. In dieser Arbeit stelle ich Deep Lagrangian Network (DeLaN) vor, einen Deep Learning Ansatz für inverse Dynamik Modelle. Die Neuheit an DeLaN ist die Einbindung einer physikalischen Voraussetzung, der partiellen Lagrange-Euler Differentialgleichung. Damit enkodiert DeLaN physikalisches Wissen und erlaubt gleichzeitig die Nutzung von Vorteilen des klassischen Machine Learnings, wie Ende-zu-Ende Training basierend auf einzelnen Messwerten. Ich vergleiche meinen Ansatz in verschiedenen Experimenten zu einem klassischen Feed-Forward Neural Network, um die Anwendbarkeit DeLaNs auf Online Learning von inversen Dynamik Modellen zu zeigen. Außerdem stelle ich die Vorteile und Einschränkungen meines Ansatzes dar, die sich aus den Eigenschaften der eingebundenen physikalischen Voraussetzung ergeben. Kurz gesagt, DeLaN ist in der Lage in Echtzeit zu lernen und einen echten Roboter zu steuern und es bietet dabei die Fähigkeit zu neuen, unbekannten Aufgaben zu extrapolieren.

Acknowledgments

I would first like to thank my thesis advisor Michael Lutter of the Intelligent Autonomous Systems (IAS) group at TU Darmstadt. He supported me in this work by giving great feedback and ideas. Without his valuable contribution this thesis in its current shape would not have been possible. Furthermore, I would like to thank Prof. Dr. Jan Peters, head of IAS, for making this thesis possible.

I would also like to acknowledge Matthias Seubert and Felix Weidinger as the second readers of this thesis. I am grateful for your very valuable comments and honest feedback.

Finally, I must thank my parents, my sister and my fiancée for providing me with unfailing support and continous encouragement throughout the years of my study and the time of this thesis in particular. Without them this accomplishment would not have been possible.

Thank you all.
Christian Ritter

Contents

Figures

List of Figures

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
DeLaN	Deep Lagrangian Network
dof	degrees of freedom
FF-NN	feed-forward neural network
MLP	multilayer perceptron
MSE	mean squared error
PDE	partial differential equation
ReLU	Rectified Linear Unit
RNE	recursive Newton-Euler algorithm
w.r.t.	with respect to

List of Symbols

Notation	Description
$\mathbf{q}$	vector of joint positions of a robot
$\dot{\mathbf{q}}$	vector of joint velocities of a robot
$\ddot{\mathbf{q}}$	vector of joint accelerations of a robot
$\boldsymbol{\tau}$	vector of joint torques or forces of a robot
$\mathbf{a}_i$	activations of the i-th layer of a neural network
$\mathbf{b}_i$	biases of the i-th layer of a neural network
$\mathbf{h}_i$	outputs of the i-th layer of a neural network
$\mathbf{W}_i$	weights of the i-th layer of a neural network
$\boldsymbol{\theta}$, $\boldsymbol{\psi}$	vectors of parameters of a neural network

List of Operators

Notation	Description	Operator
$\hat{\cdot}$	approximation of a function	$\hat{\cdot}$
g	non-linear activation function of a neural network	$g(\bullet)$

1 Introduction

In recent years deep neural networks have led to great advances in many fields of machine learning. From already being the state-of-the-art in almost all computer vision applications [1, 2] to becoming more and more prominent in many other fields like reinforcement learning [3], speech recognition [4], or artificial intelligence in games [5].

One field of research where deep learning has yet to prove itself is model learning. That is the utilization of machine learning techniques to approximate a physical model. Learning models can be used for a variety of cases, e.g., for controlling a robot. Given the success of deep learning in other fields, applying it to model learning promises good results. Though, currently most engineers prefer classical off-the-shelf modeling as it ensures physical plausibility - at a high cost of precise measurements[1] and engineering effort. Finding a simple yet efficient way to derive models that perform on a similar or even higher level than their analytical counterpart could save time and money.

However, to enable learn models with deep neural networks, three major problems have to be faced. First, a problem in many machine learning applications is that, in order to train a sufficient model, huge amounts of labeled training data are required. This is especially important for deep learning. Depending on the task this data is more widely available (e.g., in image classification existing datasets can be used to pre-train a model [7]) while it is hard to obtain elsewhere. Learning the dynamics model of a real robot requires to collect training data. Thus, in order to gather data points, trajectories have to be executed on the (real) robot for which a model should be learned [8, 9]. Given the infinite possible trajectories it is hard to find a set of training data that is both, large enough to lead to a good model and small enough not to exceed memory and time limitations. Moreover, if the task for which the model should be learned is unknown at training time it becomes particularly hard to find a sufficient training dataset. Second, the approximations learned in model learning are data-based and, thus, are unaware of any underlying physics. This lack of physical plausibility may be one reason why engineers are currently not using model learning. While the performance of learned models may be comparable to analytical models they do not ensure robustness and extrapolation to unknown samples. It is, however, not a trivial task to incorporate physics into a machine learning model while keeping the benefit of end-to-end training which is provided by the data-based approaches. Third, (deep) neural networks are often referred to as being intransparent or "black-boxes". This is due their many layers forming a complex architecture that is hard to understand or visualize. Although their complex structure is one of the reasons of their successes in the past, it is desirable to increase the understanding of a learned neural network. While some efforts have been made to make deep neural networks more accessible [10, 11], it remains a continuing research effort.

In order to overcome those issues I propose the following solutions. First, as shown by various authors [12, 13, 14], online learning can be used to obtain train samples on-the-fly. This thesis utilizes online learning by collecting data points while the robot is executing its trajectories and using those to train and update the model. Thereby, it does not only overcome the need of pre-collected data but also collects data that is relevant for the task at hand. Moreover, I introduce a system architecture featuring an asynchronous training process. This ensures the applicability of the learned model in real-time where high prediction rates, much faster than training is currently possible, are required. Second, I propose the combination of a standard feed-forward neural network (FF-NN) and a physics prior based on Lagrangian mechanics that is called Deep Lagrangian Network (DeLaN). DeLaN enables the use of neural networks while profiting from knowledge already present. Furthermore, it can be shown that the proposed method could be used to incorporate a variety of other partial differential equations (PDEs) into a neural network. The resulting network architecture can be trained end-to-end which allows the use of standard optimization algorithms that are already widely used in deep learning. This means that DeLaN can still be used in the same way as any other model learning approach, i.e., it does not require any additional data. Moreover, DeLaN is able to model the inverse dynamics model of any mechanical system without additional knowledge about the system beside its degrees of freedoms (dofs). It is, thus, a general means to model a physical system. Third, by enforcing physical plausibility via incorporating prior knowledge some transparency is added to the model as it becomes more clear which part of the function is learned and how the learned approximation fits into the final model. However, it still allows to benefit from the modeling power of deep neural networks. This may increase the acceptance of learned models in areas where reliability is important, e.g., in industrial applications or in human-robot interaction tasks.

In this thesis I conduct multiple experiments to showcase the performance of DeLaN in comparison to both an ana-

[1] Highly precise models usually require taking the physical system apart and measuring the separated pieces [6].

lytical model as well as a classical FF-NN. DeLaN is tested in a real-time scenario on a simulated robot as well as the real seven dofs robot Barrett WAM. The experiments show that DeLaN performs on a comparable level with the analytical model maintaining a good extrapolation to unseen samples. Furthermore, it can be trained like any other neural network in an end-to-end fashion without any adaptation to data collection or the optimization algorithms.

During the work on this thesis a paper was created and submitted to ICLR [15]. It is currently under a double blind review process.

The thesis is structured as follows. Chapter 2 explains the background this work is built on and showcases important related work. Chapter 3 contains the mathematical derivations required to incorporate a PDE into a neural network as well as the description of the architecture of DeLaN. The experiments and their results are described in Chapter 4 followed by a discussion about the idea and approach in Chapter 5. Chapter 6 concludes my thesis by summarizing the results of this work and highlighting future work to improve DeLaN.

2 Background and Related Work

In this chapter I highlight the background this work is built on as well as relevant related work. Section 2.1 explains the basics of model learning and inverse dynamics control, Section 2.2 covers deep learning, and Section 2.3 closes with related works.

2.1 Model Learning and Inverse Dynamics Control

Nguyen-Tuong et al. describe models as the the basic underlying mechanisms of the world. In robotics, for example, models are required for kinematics, inverse dynamics control, autonomous navigation and many more applications. As good models are hard to obtain analytically due to a lack of physical knowledge, acquiring them with techniques from machine learning has been an upcoming field of research called model learning. [13] This thesis focuses mainly on learning inverse dynamics models. A great overview of the basics and challenges of model learning can be found in a survey by Nguyen-Tuong et al. [13].

Nguyen-Tuong et al. name four types of models: forward, inverse, mixed, and multi-step prediction models. Forward models directly represent a mapping from action to state, i.e., they represent the state-transfer function of the system. Opposite to that, inverse models describe the action required to reach a desired state. Mixed models are a combination of the two aforementioned models that can be used to tackle ill-posed inverse problems by introducing a forward model that encodes additional information. Finally, multi-step prediction models are similar to forward models although they do not only predict the state-transition for the next step but rather for a finite set of future steps. [13]

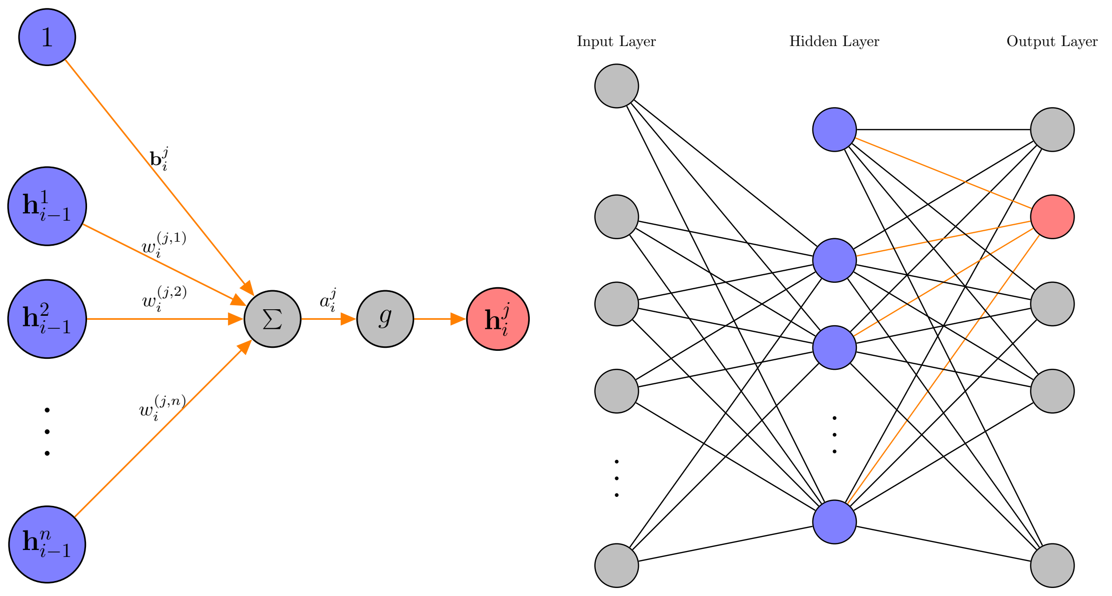
Three major modeling architectures exist. First, direct modeling is a straightforward approach where a direct mapping from input to output is learned. This is also the modeling architecture used in this work where the inputs are defined as the desired state and the outputs are the torques required to reach that state [16]. Inverse dynamics can be directly modeled as the problem is well-defined. Second, indirect modeling is used for ill-posed problems like inverse kinematics. Here, a direct relationship from input to output is not given. A prominent example of indirect modeling is feedback error learning [17, 18]. It uses the error of a feedback controller to learn the feed-forward model and has also been applied to inverse dynamics learning in the past. One advantage of using the feedback error is that it measures the performance on the task rather than on the training samples. Third, distal teacher learning is also used for ill-posed problems but utilizes a forward model to steer the teaching of an inverse model [19]. By minimizing the error of a unique forward model the non-uniqueness of the inverse model can be mitigated.

Craig gives a summary of mechanics in robotics including the formulation of inverse dynamics for a robot [20]. In general, models describing system dynamics, i.e., the coupling of system-input $\boldsymbol{\tau}$ and state $\mathbf{q}$, are essential for model-based control approaches [21]. Depending on the control approach, the control law relies either on the forward dynamics model f, a mapping from control input to the change of system state, or on the inverse dynamics model f^{-1}, a mapping from system change to control input, i.e.

$$f(\mathbf{q},\, \dot{\mathbf{q}},\, \boldsymbol{\tau}) = \ddot{\mathbf{q}}, \qquad f^{-1}(\mathbf{q},\, \dot{\mathbf{q}},\, \ddot{\mathbf{q}}) = \boldsymbol{\tau}.$$

Inverse dynamics control [22] is a prominent example of utilizing the inverse model to compensate system dynamics, while model-predictive control [23] or optimal control [24] use the forward model to plan the control input.

In contrast to model learning, classical mechanics use analytical models. Those analytical models are often obtained using parameter estimation methods on pre-defined physical models which optimize the model parameters from measured data either in a special parameter estimation process [25] or online using adaptive control [26]. A prominent example to derive a model from known parameters is the recursive Newton-Euler algorithm (RNE) [27]. In order to achieve precise models with this techniques it is necessary to know certain properties of the robot, e.g., inertia tensors or centers of mass. To gain this information either complex simulations are required or, for more precise models, the system has to be taken apart and each part has to be measured individually [6]. This has to be repeated for each individual mechanical system.



(a) A single neuron **(b)** A multilayer perceptron

Figure 2.1.: Schematic drawing of a single neuron (a) and a multilayer perceptron with one hidden layer (b).
(a) The shown neuron represents the j-th neuron of the i-th layer of a multilayer perceptron. It takes all outputs of the previous layer as input and multiplies each with a respective weight. The weighted inputs are summed, resulting in the activation a_i^j. A bias b_i^j is also added to this activation. The non-linear function g is applied, generating the neuron output h_i^j. Thus, the full equation of a single neuron is given by $h_i^j = g\left(\mathbf{w}_i^j \mathbf{h}_{i-1} + b_i^j\right)$.
(b) A fully connected multilayer perceptron with one hidden layer. A single neuron is highlighted by the same color coding as seen in (a). Each layer except for the output layer contains a bias input with value 1 weighted by the respective bias parameter b_i^j. While the neurons of one layer are independent, each one relies on all neurons of the previous layer.

2.2 Deep Learning

Deep learning is a class of machine learning algorithms that use multiple layers of neurons inspired by those of the human brain. In general, the goal of any machine learning algorithm is to learn a mapping from a (multidimensional) input space X to a (multidimensional) output space Y. Models are learned as an approximation from samples $\mathbf{x} \to \mathbf{y}$ of the true function by means of optimization. From their first appearance in the 1940s, neural networks have shown great success in various fields of machine learning. An overview of the history of deep learning is given by Schmidhuber [28].

The most simple form of deep neural network is the Multilayer Perceptron (MLP) or feed-forward neural network (FF-NN). It consists of multiple layers where each layer is a set of neurons with multiple inputs that are weighted and summed up, i.e., an affine transformation is performed. Afterwards, a non-linear activation function is applied and the result is passed to the neurons of the next layer. The i-th layer of a FF-NN is defined as

$$\mathbf{h}_i = g_i\left(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i\right) = g_i\left(\mathbf{a}_i\right), \tag{2.1}$$

where $\mathbf{W}_i$ are the weights applied to the inputs of the previous layer $\mathbf{h}_{i-1}$, $\mathbf{b}_i$ are biases and g_i is the non-linear activation function. $\mathbf{a}_i$ is called the activation of the neuron. [29]

A schematic drawing of a single neuron can be seen in Figure 2.1a. It shows the computational graph of the affine transformation and the non-linearity. Figure 2.1b shows a multilayer perceptron with one hidden layer. The single neuron of Figure 2.1a is highlighted by color.

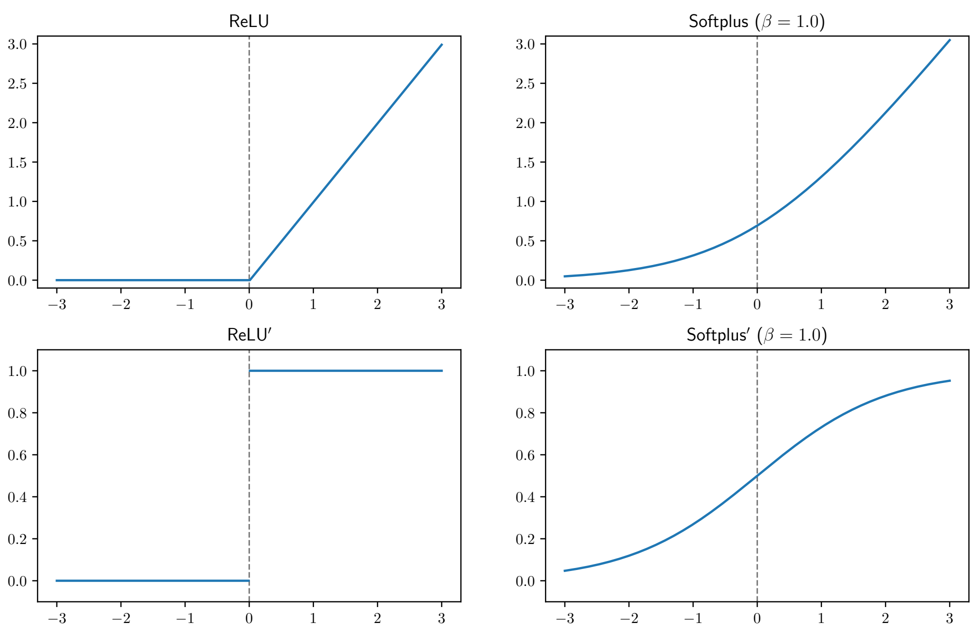


Figure 2.2.: Activation functions and their derivatives. On the left, the Rectified Linear Unit (ReLU) and its derivative are plotted. On the right, Softplus, with parameter $\beta = 1.0$, and its respective derivative is shown. Both activation functions result in a positive output that has no upper bound. It can be seen that Softplus resembles a smooth approximation of the ReLU function.

In this work I use two different activation functions: Rectified Linear Unit (ReLU) [30] and Softplus [31]. They are defined as

$$\mathrm{ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}, \qquad \mathrm{Softplus}(a) = \frac{1}{\beta} \log\left(1 + \exp(\beta a)\right).$$

In Softplus, β is an arbitrary parameter, although, the higher β the closer Softplus approximates ReLU. Figure 2.2 shows the plots of both activation functions and their derivatives for activations $a = [-3, 3]$. It can be seen that Softplus has a similar, but smoother, shape as ReLU.

When training a neural network the weights and biases have to be learned, i.e., a loss function is optimized w.r.t. the parameters $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$. This is often done using back-propagation [32] where the gradient of an error function is passed backwards through the network. Multiple optimization algorithms have been proposed in the past, e.g., ADAM [33], AdaGrad [34], or RMSProp [4]. In today's frameworks, the computation of the gradients is usually done using the reverse of the automatic differentiation [35], often called Autograd. An exhaustive overview of neural networks and deep learning is given by Haykin [36] and Goodfellow et al. [29].

2.3 Related Work

In this section I highlight the decisions made by related works compared to my approach.

When learning models, mostly standard machine learning techniques are applied to fit either the forward- or inverse-model to sensor data. Often used algorithms are Linear Regression [9, 37], Gaussian Mixture Regression [38, 39], Gaussian Process Regression [40, 12, 41], Support Vector Regression [42, 43], feed-forward- [44, 45, 46, 47] or recurrent neural networks [8]. Only a few exceptions incorporate prior knowledge into the learning problem. Atkeson et al. used the Newton-Euler formalism to derive physics features such that learning of the inverse dynamics model simplifies to linear regression [48]. Similarly, Ledezma et al. encoded the equations describing the analytical inverse dynamics model, derived using the Newton-Euler formalism, as network and trained the network using end-to-end training [46]. Sanchez-Gonzalez et al. used the graph representation of the mechanical system as structured input [47]. However, those approaches are specific to the task at hand and do not extend easily to various systems as the approach I propose does. Hu et al. summarize multiple approaches from different fields of study into the general idea of incorporating prior knowledge into a neural network. They call it generalized-constrained neural network (GCNN). Their work states various ways how and where knowledge could be encoded in a neural network. Following their definition, my approach would also be a GCNN. [49] However, they do only describe the idea of constraining a neural network on a very basic level and are not mentioning PDEs.

With DeLaN I follow the line of structured learning problems but use a more general formulation. Rather than requiring knowledge of the physical system to construct a graph or derive the Newton-Euler approach, DeLaNs are identical for all mechanical systems and do not require specific knowledge of those. Only the system state and input is specific to each system but neither the topology nor the optimization procedure.

In related works, the learned models were either trained using data collected offline [9, 8] or online [12]. The online data collection does not require collecting samples prior to training and provides better models as the acquired data distribution depends on the control law and task [12]. Therefore, I also utilize online learning with the additional advantage of mitigating the problem of persistent excitation [50, 25]. Furthermore, being able to train online makes DeLaN a more versatile approach compared to slow training models like Gaussian Process Regression.

When evaluating the model performance, either the mean squared error (MSE) [9, 8, 46] on test data or the tracking error of desired trajectories [12, 47] was used. The tracking error is the relevant performance indicator as the MSE exaggerates model performance [51, 52] based on samples and a better MSE does not necessarily imply a better tracking error. Thus, within my evaluation I assess the performance using the tracking error as it measures the performance w.r.t. the task that is performed. In addition to most previous works, I also strictly limit all model predictions to real-time and use untrained models, i.e., the models are randomly initialized and must learn during the experiment. It should be noted that training is performed using the MSE.

The combination of PDEs and neural networks has also previously been investigated in literature. Early on Lagaris et al. proposed to learn PDE solutions using neural networks [53, 54] and currently this topic is being rediscovered by various authors [55, 56, 57]. Most research focuses on using machine learning to overcome the limitations of PDE solvers. Sigarno et al., for example, proposed the Deep Galerkin method to solve a high-dimensional PDE from scattered data [56]. Similar to this work, Raissi et al. took the opposite standpoint of using the knowledge of the specific PDE to structure the learning problem and achieve lower sample complexity [58]. One approach by Sahoo et al. uses deep learning to discover equations that relate from input to output of a system, i.e., they discover PDEs by learning a model [59].

I follow the same motivation of utilizing information from a PDE. But rather than explicitly solving the PDE, DeLaN only uses the structure of the PDE to guide the learning problem of inferring the equations of motion. Thereby, the PDE is only implicitly solved. In addition, the proposed approach uses a different encoding of the partial derivatives, which achieves the efficient computation within a single feed-forward pass, enabling the application of DeLaN within real-time control loops.

3 Approach

In this chapter I showcase the derivations and network architecture required to encode Lagrangian Mechanics as physics prior into a neural network. I keep all derivations on a general level as the approach is applicable to any physical system. In the experiment chapter (Chapter 4) the approach is extensively evaluated on inverse dynamics control of a robot. Starting in Section 3.1 I highlight the basic equations of Lagrangian Mechanics that are the basis upon which the approach is built. Section 3.2 states the learning problem and requirements that have to be met in order to encode a PDE into a neural network. Section 3.3 introduces DeLaN and shows how it can be built from a standard FF-NN. The chapter is concluded in Section 3.4 by a description of the online learning system used in this thesis.

3.1 Lagrangian Mechanics

Describing the equation of motions for mechanical systems has been extensively studied in the past. Therefore, various formalisms to derive these equations exist. Among the most prominent are Newtonian, Hamiltonian, and Lagrangian Mechanics. The proposed network architecture of this work, DeLaN, is based on Lagrangian Mechanics [60, 22, 61], more specifically the Lagrange-Euler formulation with non-conservative forces and generalized coordinates [60]. Generalized coordinates are coordinates that uniquely define the system configuration. The Lagrange-Euler formalism defines the Lagrangian L as a function of generalized coordinates $\mathbf{q}$ describing the complete dynamics of a given system. It should be noted that the Lagrangian is not unique and every L, which yields the correct equations of motions is valid. In general, the Lagrangian is chosen to be

$$L = T - V, \tag{3.1}$$

where T is the kinetic energy and V is the potential energy of the system. The kinetic energy T can be computed for all choices of generalized coordinates using $T = 1/2\,\dot{\mathbf{q}}^\intercal \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}$, where $\mathbf{H}(\mathbf{q})$ is the inertia matrix which has to be symmetric and positive definite [22]. The positive definiteness ensures that all non-zero velocities lead to positive kinetic energy. Applying the calculus of variations yields the Euler-Lagrange equation with generalized forces (or torques) $\boldsymbol{\tau}$ described by

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}_i} - \frac{\partial L}{\partial \mathbf{q}_i} = \boldsymbol{\tau}_i. \tag{3.2}$$

Substituting L using Equation (3.1) into Equation (3.2) and assuming $\partial V/\partial \mathbf{q} = \mathbf{g}(\mathbf{q})$ and $\partial V/\partial \dot{\mathbf{q}} = \mathbf{0}$ yields

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^\intercal \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^\intercal + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}. \tag{3.3}$$

Within engineering disciplines Equation (3.3) is further abstracted to

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}. \tag{3.4}$$

In this equation $\mathbf{H}$ is the inertia or mass matrix, $\mathbf{c}$ abstracts the forces generated by the Centripetal and Coriolis forces and $\mathbf{g}$ corresponds to the gravitational force [61].

3.2 Incorporating Lagrangian Mechanics into Neural Networks

Starting from the Lagrange-Euler PDE (Equation (3.3)), traditional engineering approaches would estimate $\mathbf{H}(\mathbf{q})$, $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$, $\mathbf{g}(\mathbf{q})$ from approximated or measured masses, lengths, and moments of inertia. These can be obtained by either measuring out the system or by complex simulations, e.g., using the finite element method. On the other hand, most traditional model learning approaches would ignore any physical structure and learn the inverse dynamics model directly from data.

The architecture I propose, DeLaN, bridges the gap between the two aforementioned approaches by incorporating the structure introduced by the Lagrange-Euler PDE into the learning problem. However, DeLaN can be trained like

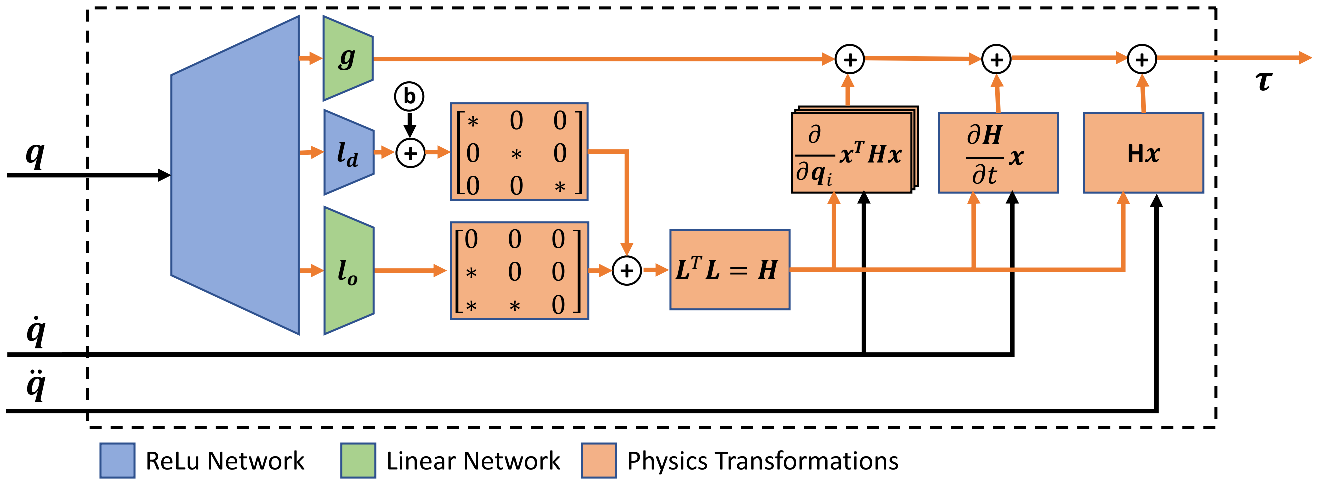

q
$\dot{q}$
$\ddot{q}$
g
l_d
l_o
b
$\begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 0 \\ * & 0 & 0 \\ * & * & 0 \end{bmatrix}$
$L^T L = H$
$\frac{\partial}{\partial q_i} x^T H x$
$\frac{\partial H}{\partial t} x$
Hx
τ
🟦 ReLu Network
🟩 Linear Network
🟧 Physics Transformations

Figure 3.1.: The computational graph of Deep Lagrangian Network (DeLaN). Shown in blue and green are the neural networks with the three separate heads computing $\mathbf{g}(\mathbf{q})$, $\mathbf{l}_d(\mathbf{q})$, $\mathbf{l}_o(\mathbf{q})$, where $\mathbf{l}_d$ and $\mathbf{l}_o$ are the vectorized form of the diagonal and lower part of $\mathbf{L}$ respectively. $\mathbf{l}_d$ uses ReLU activations to ensure that the diagonal of $\mathbf{L}$ is non-negative and the addition of a small bias b makes the diagonal positive. Hence, H is positive definite. The orange boxes correspond to reshaping operations and the derivatives contained in the Lagrange-Euler equation. For training, the gradients are backpropagated through all orange paths. It can also be seen that the networks share the same base as all heads depend on the same mechanical system.

any classical machine learning model in an end-to-end fashion. More precisely, DeLaN approximates the inverse dynamics model of any physical system by representing the unknown functions $\mathbf{g}(\mathbf{q})$ and $\mathbf{H}(\mathbf{q})$ as FF-NNs. Rather than modeling $\mathbf{H}$ directly, it is learned via its lower triangular matrix $\mathbf{L}$, as explained in Section 3.2.1. Therefore, $\mathbf{g}(\mathbf{q})$ and $\mathbf{H}(\mathbf{q})$ are described by

$$\mathbf{H}(\mathbf{q}) = \hat{\mathbf{L}}(\mathbf{q}\,;\,\boldsymbol{\theta})\,\hat{\mathbf{L}}(\mathbf{q}\,;\,\boldsymbol{\theta})^{\mathsf{T}}, \qquad \mathbf{g}(\mathbf{q}) = \hat{\mathbf{g}}(\mathbf{q}\,;\,\boldsymbol{\psi}), \tag{3.5}$$

where $\hat{\,}$ refers to the approximation of the true function by a FF-NN and $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$ are the respective network parameters. The parameters can be obtained by minimizing the violation of the physical law described by Equation (3.3). This results in the following optimization problem to find optimal network parameters $\boldsymbol{\theta}^*$ and $\boldsymbol{\psi}^*$.

$$\boldsymbol{\theta}^*,\ \boldsymbol{\psi}^* = \underset{\boldsymbol{\theta},\boldsymbol{\psi}}{\arg\min}\ \ \ell\left(\hat{f}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\,;\,\boldsymbol{\theta}, \boldsymbol{\psi}),\ \boldsymbol{\tau}\right), \tag{3.6}$$

$$\text{with} \quad \hat{f}^{-1}(\mathbf{q},\, \dot{\mathbf{q}},\, \ddot{\mathbf{q}}\,;\,\boldsymbol{\theta},\,\boldsymbol{\psi}) = \hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}\,\ddot{\mathbf{q}} + \frac{d}{dt}\left(\hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}\right)\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^{\mathsf{T}}\,\hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}\,\dot{\mathbf{q}}\right)\right)^{\mathsf{T}} + \hat{\mathbf{g}}, \tag{3.7}$$

$$\text{s.t.}\ \ 0 < \mathbf{x}^{\mathsf{T}}\ \hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}\,\mathbf{x}, \qquad \forall\ \mathbf{x} \in \mathbb{R}^n_0. \tag{3.8}$$

$\hat{f}^{-1}$ is the inverse dynamics model and ℓ can be any differentiable loss function. The samples used for training are the joint state, i.e., joint positions, velocities, and accelerations $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$, and the torques $\boldsymbol{\tau}$ that caused the states.
The computational graph of $\hat{f}^{-1}$ is shown in Figure 3.1. As can be seen, $\hat{\mathbf{L}}$ and $\hat{\mathbf{g}}$ both depend on a shared network. Only a final layer, in this work also called network head, is different for the learned functions. This keeps the network architecture on a smaller scale. Parameters are shared between $\mathbf{L}$ and $\mathbf{g}$ as both rely on the same physical system. As the network outputs a one-dimensional vector, reshaping operations are applied to obtain $\mathbf{g}$ and $\mathbf{g}$ and matrix calculus combines them with the inputs resulting in the torque described in Equation (3.7).

From the formulation of Equation (3.5) a property of the approach can be concluded. Neither $\hat{\mathbf{L}}$ nor $\hat{\mathbf{g}}$ depend on $\dot{\mathbf{q}}$ or $\ddot{\mathbf{q}}$. Hence, the obtained models should, within limits, generalize to arbitrary velocities and accelerations. Moreover, this results in a smaller input space compared to a FF-NN which depends on the full state.

In addition, the learned model can be reformulated and used as a forward model. Solving Equation (3.7) for $\ddot{\mathbf{q}}$ yields the forward model described by

$$\hat{f}(\mathbf{q},\, \dot{\mathbf{q}},\, \boldsymbol{\tau}\,;\,\boldsymbol{\theta},\,\boldsymbol{\psi}) = \left(\hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}\right)^{-1}\left(\boldsymbol{\tau} - \frac{d}{dt}\left(\hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}\right)\dot{\mathbf{q}} + \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^{\mathsf{T}}\,\hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}\,\dot{\mathbf{q}}\right)\right)^{\mathsf{T}} - \hat{\mathbf{g}}\right), \tag{3.9}$$

where $\hat{\mathbf{L}}\,\hat{\mathbf{L}}^{\mathsf{T}}$ is guaranteed to be invertible due to the positive definite constraint (Equation (3.8)). This means that DeLaN could also be used in cases where a forward model is required, e.g., optimal control, even if it is only trained in an inverse dynamics use case.

One remaining issue is that the optimization problem of Equation (3.6) cannot be solved directly due to the ill-posedness of the Lagrangian L not being unique. This can be seen as the Lagrange-Euler PDE is invariant to linear transformation. Hence, any Lagrangian $L' = \alpha L + \beta$ solves the Lagrange-Euler PDE if α is non-zero and L is a valid Lagrangian. This problem can be mitigated by adding an additional penalty term to Equation (3.6), i.e.,

$$\boldsymbol{\theta}^*,\ \boldsymbol{\psi}^* = \underset{\boldsymbol{\theta},\boldsymbol{\psi}}{\arg\min}\ \ \ell\left(\hat{f}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\,;\, \boldsymbol{\theta}, \boldsymbol{\psi}),\ \ \boldsymbol{\tau}\right) + \lambda\ \Omega(\boldsymbol{\theta},\ \boldsymbol{\psi}), \tag{3.10}$$

where Ω is the L_2-norm of the network weights and λ is a parameter adjusting the impact of the penalty term. In the domain of deep learning this is also known as regularization.

The optimization problem of Equation (3.10) is built on two assumptions that have to hold in order to solve the problem with a gradient based end-to-end approach. First, the matrix $\mathbf{H}(\mathbf{q})$ is symmetric and positive definite for all network parameters. Second, the derivatives contained within the PDE can be computed in closed form using a single feed-forward pass.

In the following sections, it is proven that both necessary conditions hold for the proposed architecture. First a network architecture is introduced, which ensures the symmetry and positive definiteness of $\mathbf{H}$ for all parameters (Section 3.2.1). Afterwards, the derivatives within the Lagrange-Euler PDE are derived in closed form (Section 3.2.2).

3.2.1 Symmetry and Positive Definiteness of $\mathbf{H}$

Ensuring the symmetry and positive definiteness of $\mathbf{H}$ is essential as this constraint enforces positive kinetic energy for all non-zero velocities. In addition, the positive definiteness makes $\mathbf{H}$ invertible and, thus, the obtained model can be used as forward model.

By representing the matrix $\mathbf{H}$ as the product of a lower-triangular matrix symmetry and positive semi-definiteness is secured. It also reduces the number of parameters of the respective network head by roughly a half. Furthermore, $\mathbf{H}$ is guaranteed to be positive definite if the diagonal elements of $\mathbf{L}$ are positive. To achieve this, DeLaN uses two different heads for representing $\mathbf{L}$. One, modeling the diagonal $\mathbf{L}_d$, uses a non-negative activation function, e.g., ReLu or Softplus, and adds a small, positive scalar b to each elements to ensure that they are non-zero. Thereby, the eigenvalues of $\mathbf{H}$ are lower bounded by b^2. The other one, $\mathbf{L}_l$, with a linear activation, models the elements below the diagonal of $\mathbf{L}$. The positive diagonal also makes $\mathbf{L}$ invertible.

Figure 3.1 shows how $\mathbf{L}$ is obtained by reshaping and adding the one-dimensional outputs $\mathbf{l}_d$ and $\mathbf{l}_l$ of the network.

3.2.2 Derivatives of $\mathbf{L}$

The inverse dynamics formulation of Equation (3.7) requires two derivatives of $\mathbf{L}$. As $\mathbf{L}$ is modeled by a neural network the derivatives of the network form the derivative of $\mathbf{L}$. Recalling the definition of the i-th layer of a neural network from Section 2.2 (Equation (2.1))

$$\mathbf{h}_i = g_i\left(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i\right) = g_i\left(\mathbf{a}_i\right),$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are the learnable parameters of that layer. Using the chain rule, the derivative of the i-th layer w.r.t. $\mathbf{q} = \mathbf{h}_0$ is

$$\frac{\partial \mathbf{h}_i}{\partial \mathbf{q}} = \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \frac{\partial \mathbf{h}_{i-1}}{\partial \mathbf{q}}, \qquad \text{with } \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \begin{bmatrix} \frac{\partial \mathbf{h}_i^0}{\partial \mathbf{h}_{i-1}^0} & \cdots & \frac{\partial \mathbf{h}_i^0}{\partial \mathbf{h}_{i-1}^{n_{i-1}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{h}_i^{n_i}}{\partial \mathbf{h}_{i-1}^0} & \cdots & \frac{\partial \mathbf{h}_i^{n_i}}{\partial \mathbf{h}_{i-1}^{n_{i-1}}} \end{bmatrix} \quad \text{and } \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}^k} = g'\left(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i\right) * \mathbf{W}_i \mathbf{e}_k,$$

$$\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \text{diag}\left(g'(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i)\right) \mathbf{W}_i, \tag{3.11}$$

$$\frac{\partial \mathbf{h}_N}{\partial \mathbf{q}} = \frac{\partial \mathbf{h}_N}{\partial \mathbf{h}_{N-1}} \frac{\partial \mathbf{h}_{N-1}}{\partial \mathbf{h}_{N-2}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{q}}. \tag{3.12}$$

g' resembles the derivative of the non-linear activation function, $\mathbf{e}_k$ is the k-th unit vector, and $(*)$ denotes an element-wise multiplication. As can be seen in Equation (3.12) the derivative can be passed through the network by calculating the derivative of each layer w.r.t. the previous layer.
Similarly, the derivative of the i-th layer w.r.t. time is

$$\frac{d\mathbf{h}_i}{dt} = \frac{\partial \mathbf{h}_i}{\partial \mathbf{q}} \frac{d\mathbf{q}}{dt} = \frac{\partial \mathbf{h}_i}{\partial \mathbf{q}} \dot{\mathbf{q}}. \tag{3.13}$$

This shows that only the derivative of Equation (3.12) is required to calculate both derivatives needed in my approach, as $\dot{\mathbf{q}}$ is available from the network input.

These derivatives can now be applied to $\mathbf{L}$. Let $\mathbf{l}$ be the vectorized form of the non-zero entries of $\mathbf{L}$, i.e.,

$$\mathbf{l} = (l_{11}, l_{21}, l_{22}, ..., l_{n,n}), \quad \text{where } \mathbf{L} = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}.$$

Note that all following equations can also be applied to the split heads $\mathbf{l}_d$ and $\mathbf{l}_o$, see Figure 3.1, even though they are only derived for $\mathbf{l}$. For $\mathbf{l}$ being the final layer $\mathbf{h}_N$ of the network the derivative w.r.t. time can be calculated using Equation (3.13)

$$\frac{d}{dt}\mathbf{l} = \frac{d}{dt}\mathbf{h}_N = \frac{\partial \mathbf{h}_N}{\partial \mathbf{q}} \dot{\mathbf{q}} = \frac{\partial \mathbf{l}}{\partial \mathbf{q}} \dot{\mathbf{q}}. \tag{3.14}$$

As can be seen in Equation (3.12) the derivative w.r.t. $\mathbf{q}$ can be computed by applying the chain rule, i.e.,

$$\frac{\partial \mathbf{l}}{\partial \mathbf{q}} = \frac{\partial \mathbf{l}}{\partial \mathbf{h}_{N-1}} \frac{\partial \mathbf{h}_{N-1}}{\partial \mathbf{h}_{N-2}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{q}}. \tag{3.15}$$

The mapping between the derivatives is then defined by a reshaping operation, i.e., from $\partial \mathbf{l}/\partial \mathbf{q}$ to $\partial \mathbf{L}/\partial \mathbf{q}$. Using this property the derivatives $(d/dt)\,(\mathbf{L}\mathbf{L}^\intercal)$ and $(\partial/\partial \mathbf{q}_i)\,[\dot{\mathbf{q}}^\intercal \mathbf{L}\mathbf{L}^\intercal \dot{\mathbf{q}}]$ can be computed by reshaping the derivatives of $\mathbf{l}$.
The temporal derivative $\dot{\mathbf{H}}$,

$$\dot{\mathbf{H}} = \frac{d}{dt}\mathbf{H} = \frac{d}{dt}\left(\mathbf{L}\mathbf{L}^\intercal\right) = \mathbf{L}\frac{d\mathbf{L}}{dt}^\intercal + \frac{d\mathbf{L}}{dt}\mathbf{L}^\intercal,$$

requires $d\mathbf{L}/dt$ which is the reshaped form of $d\mathbf{l}/dt$ (Equation (3.14)).
Similarly to the previous derivation, the partial derivative of the quadratic term can be computed by the multiplication rule. That yields

$$\frac{\partial}{\partial \mathbf{q}_i}\left(\dot{\mathbf{q}}^\intercal \mathbf{L}\mathbf{L}^\intercal \dot{\mathbf{q}}\right) = \dot{\mathbf{q}}^\intercal \frac{\partial}{\partial \mathbf{q}_i}\left(\mathbf{L}\mathbf{L}^\intercal\right) \dot{\mathbf{q}} = \dot{\mathbf{q}}^\intercal \frac{\partial \mathbf{L}}{\partial \mathbf{q}_i} \mathbf{L}^\intercal \dot{\mathbf{q}} + \dot{\mathbf{q}}^\intercal \mathbf{L} \frac{\partial \mathbf{L}^\intercal}{\partial \mathbf{q}_i} \dot{\mathbf{q}},$$

whereas $\partial \mathbf{L}/\partial \mathbf{q}_i$ can be constructed using the columns of previously derived $\partial \mathbf{l}/\partial \mathbf{q}$ (Equation (3.15)).
Therefore, all derivatives included within $\hat{f}$ can be computed in closed form.

3.3 The Deep Lagrangian Network

With all required prerequisites met, I propose Deep Lagrangian Network (DeLaN). It is an adapted FF-NN able to encode the Lagrange-Euler PDE as well as other PDEs.

Following Equation (3.7), the network derivatives have to be available within the forward path. Thus, to simultaneously compute $\mathbf{l}$ and $\partial \mathbf{l}/\partial \mathbf{q}$, the forward path is extended. Using the compositional structure of $\partial \mathbf{l}/\partial \mathbf{q}$ highlighted in Equation (3.15), $\partial \mathbf{l}/\partial \mathbf{q}$ can be computed by chaining Lagrangian layers. The Lagrangian layer is an extension to the standard perceptron layer (Equation (2.1)) consisting of the affine transformation and non-linearity by adding a pathway computing $\partial \mathbf{h}_i/\partial \mathbf{h}_{i-1}$. Using Equation (3.11) the Lagrangian layer is described by

$$\mathbf{a}_i = \mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i, \qquad \mathbf{h}_i = g_i\left(\mathbf{a}_i\right), \qquad \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \text{diag}\left(g_i'(\mathbf{a}_i)\right)\mathbf{W}_i.$$

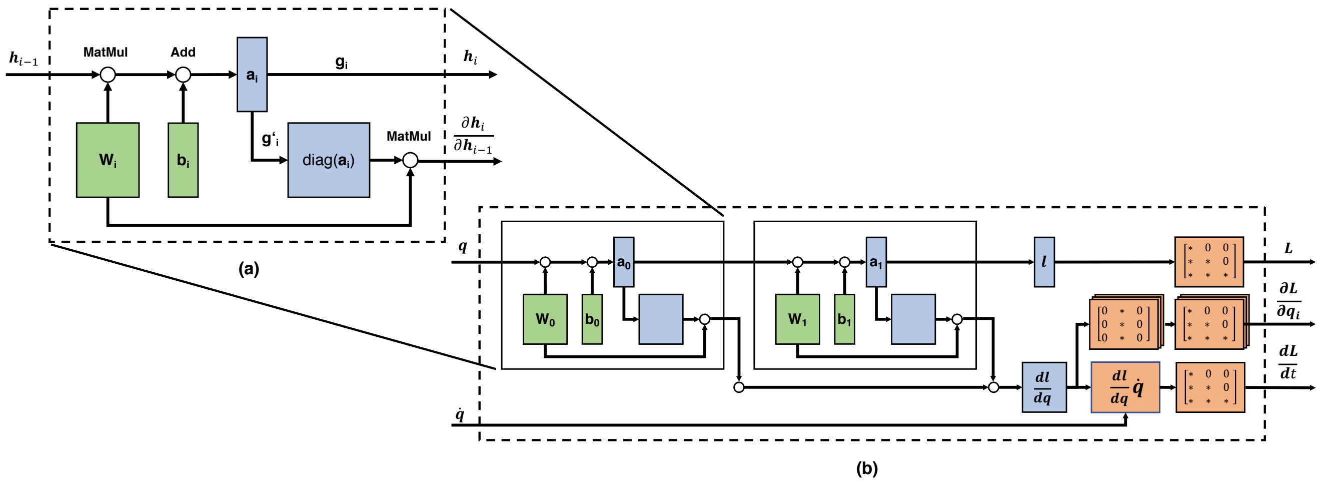


Figure 3.2.: (a) Computational graph of the Lagrangian layer. The green boxes highlight the learnable parameters. The upper computational sub-graph corresponds to the standard network layer while the lower sub-graph is the extension of the Lagrangian layer to simultaneously compute $\partial \mathbf{h}_i / \partial \mathbf{h}_{i-1}$. (b) Computational graph of the chained Lagrangian layer to compute $\mathbf{L}$, $d\mathbf{L}/dt$ and $\partial \mathbf{L}/\partial \mathbf{q}_i$ using a single feed-forward pass. Note that layers do not depend on the derivative of the previous layer as the chain rule allows independent calculation of each single derivative.

The derivatives of the activation functions used can be derived analytically. For the activation functions used in my thesis this results in

$$\mathrm{ReLU}'(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases}, \qquad \mathrm{Softplus}'(a) = \frac{\exp(\beta a)}{1 + \exp(\beta a)}.$$

Figure 2.2 shows the plot of these two derivatives for activations $a = [-3, 3]$.

Figure 3.2 (a) visualizes the computational graph of the Lagrangian Layer and Figure 3.2 (b) the resulting network structure and the reshaping operations to compute $\mathbf{L}$, $d\mathbf{L}/dt$ and $\partial \mathbf{L}/\partial \mathbf{q}_i$. I can be seen how the standard path is extended by the derivative. It also shows that the derivatives are chained but that they are computed independently per layer. The structure required for DeLaN only adds minimal computational overhead and enables the usage of automatic differentiation (Autograd) to compute the gradients w.r.t. to network weights. This means that it can be implemented and trained with any existing framework that supports Autograd. It should also be noted that the gradient passes through the derivatives' paths, i.e., the training also explicitly optimizes the derivatives.

3.4 The Online-Learning System

To be able to train DeLaN or any other machine learning model online I propose an asynchronous real-time control and training system. This architecture allows to perform real-time control even if the training takes multiple seconds. The basic system setup can be seen in Figure 3.3. It consists of two major parts, the control loop and the training process.

First, the control loop is a classical feedback loop with a feed-forward model and manages the control of the system. Thus, it receives the desired state $\mathbf{q}_d$, $\dot{\mathbf{q}}_d$, $\ddot{\mathbf{q}}_d$ and passes torques to the mechanical system. Sensor data from the resulting movement is collected and passed back to the PD-controller (feedback control). This process is running independently to ensure that the system is controlled no matter what the training process is doing. It should be noted that at the start, where no prior training has been performed, the feed-forward model's output is basically irrelevant and the executed movement is only dependent on the PD-controller. Second, the training process collects data samples from the control loop and trains new feed-forward models. The training samples consist of the measured sensor data $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and the torques passed to the system $\boldsymbol{\tau}$, i.e., a mapping $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \rightarrow \boldsymbol{\tau}$ is learned. New models are then continuously trained on all samples collected so far and the current model is passed to the control loop after each epoch of training, i.e., after the whole number of collected samples has been used for training once. Moreover, the model can be trained on any loss function. In my case the loss function is defined between the predicted and measured torques and, hence, measures the performance on the data. Note that also other loss functions, like feedback error, could be implemented which measure performance on the task.

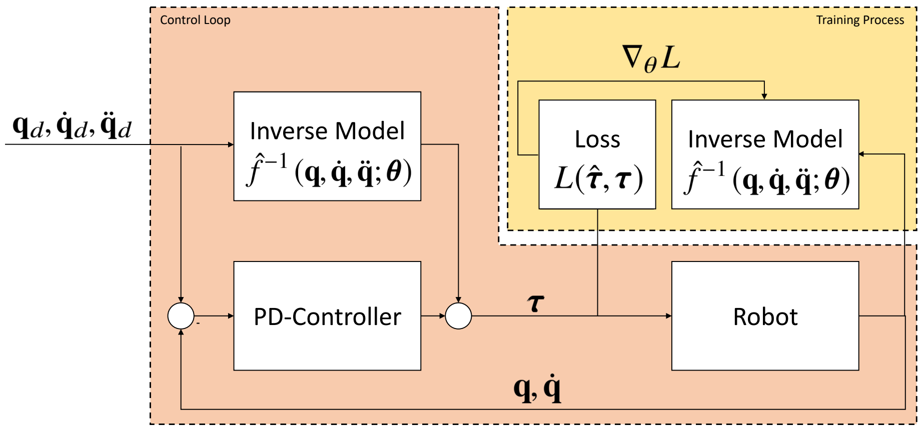

Control Loop
Training Process
$\nabla_\theta L$
$\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$
Inverse Model
$\hat{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}; \boldsymbol{\theta})$
Loss
$L(\hat{\boldsymbol{\tau}}, \boldsymbol{\tau})$
Inverse Model
$\hat{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}; \boldsymbol{\theta})$
PD-Controller
$\boldsymbol{\tau}$
Robot
$\mathbf{q}, \dot{\mathbf{q}}$

Figure 3.3.: Real-time control loop using a PD-Controller in combination with a feed-forward torque τ_{FF} to compensate the system dynamics. τ_{FF} is generated from a learned inverse dynamics model $\hat{f}^{-1}$. The training process reads the joint states and applied torques to learn the system dynamics online. This training is performed asynchronous to enable feed-forward prediction rates of 200Hz. Once a new model becomes available the inverse model $\hat{f}^{-1}$ in the control loop is updated.

4 Experiments and Results

To demonstrate the applicability to online learning and extrapolation assets of DeLaN, several experiments are conducted on the proposed network topology. Model-based control is chosen as use case as it is a typical application of the inverse dynamics model. Tests are performed on a simulated 2-dof robot (Figure 4.1b) and the physical 7-dof robot Barrett WAM, simulated and real (see Figure 4.4b). The performance of DeLaN is evaluated using the tracking error on train and test trajectories and compared to a FF-NN and an analytical model.

The chapter is structured as follows. First, Section 4.1 describes the experimental setup. It explains the structure of the test environment and the performance measure used. Furthermore, it contains information about the datasets used as well as the baselines that DeLaN is compared to. Second, Section 4.2 contains the results of all experiments performed on the simulated 2-dof robot. Third, the results of the experiments done on the real robot, Barrett WAM, are shown in Section 4.3. Following is a section about issues regarding the random initialization of the network that were found during the execution and evaluation of the experiments (Section 4.4). Finally, the chapter concludes in Section 4.5 with a brief summary of the results.

4.1 Experimental Setup

The experiments were conducted in two basic setups. Most experiments are performed online which adds special requirements to the models and setup while some are performed offline.

In the online scenario the robot has to execute multiple desired trajectories with specified joint positions, velocities and accelerations. The system used is shown in Figure 3.3. The robot is controlled by a signal of motor torques $\boldsymbol{\tau}$ which is generated in a control loop. The control loop consists of a non-linear feed-forward controller, i.e., a low gain PD-controller, in combination with a feed-forward model. The feed-forward model $\hat{f}^{-1}(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d)$ compensates system dynamics by producing a feed-forward torque $\boldsymbol{\tau}_{ff}$. It is represented by the inverse dynamics model of the robot, i.e., either an analytical model or a learned model. Thus, the control law is described by

$$\boldsymbol{\tau} = \mathbf{K}_p\left(\mathbf{q}_d - \mathbf{q}\right) + \mathbf{K}_d\left(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}\right) + \boldsymbol{\tau}_{ff}, \quad \text{with } \boldsymbol{\tau}_{ff} = \hat{f}^{-1}(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d),$$

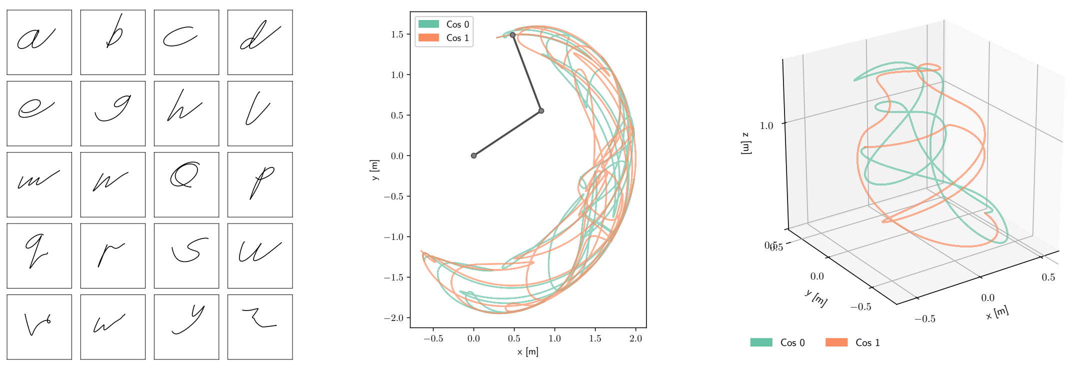
where $\mathbf{K}_p$ and $\mathbf{K}_d$ are the controller gains and $\mathbf{q}_d$, $\dot{\mathbf{q}}_d$, $\ddot{\mathbf{q}}_d$ are the desired joint positions, velocities, and accelerations. For all experiments the control frequency, i.e., the frequency with which a new torque is passed to the robot, is set to 500Hz. As the learned models require more time to make predictions the desired joint state and $\boldsymbol{\tau}_{ff}$ are updated with a frequency of $f_d = 200$Hz. Hence, the computation time of the feed-forward model is strictly limited to $T_d \leq 1/200$s.

An online experiment has two phases, the training and the testing phase. In both phases trajectories are executed. However, only in the training phase new samples are collected after the execution of a desired trajectory is finished. The samples consist of the joint states measured by the sensors and the respective torque that was passed to the robot. Once the testing phase is entered, the asynchronous training process is stopped. In order to assess the performance of the feed-forward model a trajectory tracking performance measure is used. In this work I define tracking performance as the sum of the mean squared error (MSE) evaluated at the sampling points of the reference trajectory. To be precise, for a desired trajectory $(\mathbf{x}_d)_{0,\dots,T_t}$ and a real trajectory $(\mathbf{x}_r)_{0,\dots,T_t}$, both from start time 0 to total time T_t, the MSE is defined as

$$\text{MSE}((\mathbf{x}_d)_{0,\dots,T_t}, (\mathbf{x}_r)_{0,\dots,T_t}) = \sum_{i=0}^{T_t} \left((\mathbf{x}_{d,i} - \mathbf{x}_{r,i})\right)^2 .$$

This tracking performance measure allows to evaluate and compare the performance of each feed-forward model based on their performance on the real task rather than the performance on the learning data as would be the case in a classical machine learning approach. It should be mentioned that the models, in case they are learned, are optimized on the loss between prediction and samples rather than on the tracking performance measure. This guarantees independence of training the model and controlling the robot.

In the offline learning scenarios sets of samples are collected from an online test run, either in a simulation or on the real robot. Those samples consist of input $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and target $\boldsymbol{\tau}$. The samples are then used to perform standard supervised learning. Additional information on how learning was performed is given in the description of the respective experiment.



(a) 2-dof characters **(b)** 2-dof cosine **(c)** WAM cosine

Figure 4.1.: Visualization of the task space trajectories of the datasets. (a) The character trajectories of the 2-dof robot. They contain each single stroke character. While the trajectories cover only a small portion of the 2-dof robot's task space they contain smooth and sharp turns. (b) The cosine trajectories of the 2-dof robot. They cover a wide area of the robot's task space with high velocities while being relatively smooth. (c) The cosine trajectories of the WAM. Similar to (b), the trajectories cover a wide area of the task space with high velocities.

Datasets

In this thesis I used two different datasets. The first dataset contains all single stroke characters[1] while the second data set uses cosine curves in joint space.

The characters dataset contains 20 trajectories resembling hand-writtern, single-stroke characters given in joint space. Each trajectory is spatially and temporally re-scaled to comply with the robot kinematics. To be precise, first the points at start and end of each trajectory where no movement is made, i.e., where $\dot{\mathbf{q}} = \mathbf{0}$, are cropped. Then, the total time of the trajectory is scaled to a desired total time T_t. At last, with the new time stamps and the desired joint positions the required velocities and accelerations are computed using numerical differentiation. The task space trajectories are then calculated with the forward kinematics of the robot (see Figure 4.1a). Due to the different characters, the desired trajectories contain smooth and sharp turns and cover a wide variety of different shapes but are limited to a small task space region.

In contrast, the cosine trajectories are designed to be smooth but to cover a large task space region. The trajectories are generated joint-wise, i.e., for the i-th joint the trajectory is defined by

$$q_d^i(t) = q_0^i + A^i \cos\left(2\pi{\omega^i}\beta t + \sin\left(0.5\pi t\right)\right),$$

where q_0^i is the initial joint position, A_i the amplitude, ω_i the frequency and β a velocity scaling factor. For each joint a different frequency ω_i is selected. The frequencies are chosen to be prime numbers to minimize repetition throughout a trajectory. In addition, β can be varied to compare the impact of different velocity profiles. Moreover, a sine is added within the cosine to prevent the model from being able to learn a Fourier transform rather than the true inverse dynamics model. For the 2-dof robot two trajectories with the following frequencies are created:

Name	ω^1	ω^2
"cos_0"	2	5
"cos_1"	3	7

For all trajectories the amplitude is $A = (0.3\pi, 0.5\pi)$ and the initial position is $q_0 = (-0.3, -0.1)$. Similarly, two trajectories are created for the WAM. Here, the last three joints remain still as they do not have a significant impact on the resulting trajectory and, thus, would only distort the results. The frequencies used in the cosine dataset of the WAM are as follows:

[1] The data set was created by Williams et al. [62] and is available at [63])

Name	ω^1	ω^2	ω^3	ω^4
"cos_0"	7	5	3	2
"cos_1"	7	3	5	2

The amplitude A is set to $1/6$ of the range of each respective joint, i.e., $q^i_{max} - q^i_{min}$, and q_0 is set to the middle value of the joint limits. Similarly to the characters, joint velocities and accelerations of the cosine trajectories are computed by numerical differentiation. The forward kinematics model of the robot is used to generate the task space trajectories (see Figure 4.1b and Figure 4.1c).

Baselines

In order to asses the advantages and flaws of DeLaN it has to be compared to some baselines. I chose three baselines, an analytical inverse dynamics model, a standard FF-NN and the performance of only a PD-Controller, i.e., where the output of the feed-forward model is always $\mathbf{0}$.

For the analytical models the torque is computed using the Recursive Newton-Euler algorithm (RNE) [64]. It computes the feed-forward torque $\boldsymbol{\tau}_{ff}$ using estimated physical properties of the system, i.e., the link dimensions, masses and moments of inertia. For implementations the open-source library PyBullet[2] [65] is used. The robot properties are given in URDF-files.

Both networks, DeLaN and the FF-NN, use the same architecture and ReLu non-linearities to make them comparable. They are implemented using PyTorch[3]. The optimization algorithm used is ADAM. DeLaN and the FF-NN have to learn the system dynamics starting from random initialization using the so called Xavier initialization [66]. This means that runs with different random seeds may produce different results (as investigated further in Section 4.4).

4.2 Simulated Robot Experiments

The 2-dof robot shown in Figure 4.1b is simulated using PyBullet. Three experiments were conducted. First offline training on the characters dataset to validate the approach of DeLaN. Second, the performance of the different models is evaluated online on the characers. And, third, the experiment is repeated on the cosine trajectories with different velocity profiles to assess the model's capability of extrapolation.

4.2.1 Validation of Learning the Physics Prior

In the first experiment, DeLaN is trained offline on samples of six characters and evaluated on the samples of three characters, 'a', 'd', and 'e'. Those were not part of the training set. The target torque $\boldsymbol{\tau}$ is generated by PyBullet's inverse dynamics model. The goal of this experiment is to validate the physics prior of DeLaN. DeLaN should be able to learn the individual parts of Equation (3.4), $\mathbf{H}\ddot{\mathbf{q}}$, $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{g}(\mathbf{q})$, from the super-imposed torque $\boldsymbol{\tau}$. Given the fact that PyBullet's inverse dynamics model is also based on Equation (3.4) the resulting torques should match.

To compare DeLaN and the analytical model the individual torque components of the Lagrange-Euler PDE are obtained by

$$
\begin{aligned}
\boldsymbol{\tau} &= f^{-1}(\mathbf{q},\, \dot{\mathbf{q}},\, \ddot{\mathbf{q}}),\\
\boldsymbol{\tau}_g &= f^{-1}(\mathbf{q},\, \mathbf{0},\, \mathbf{0}),\\
\boldsymbol{\tau}_c &= f^{-1}(\mathbf{q},\, \dot{\mathbf{q}},\, \mathbf{0}) - \boldsymbol{\tau}_g,\\
\boldsymbol{\tau}_H &= f^{-1}(\mathbf{q},\, \mathbf{0},\, \ddot{\mathbf{q}}) - \boldsymbol{\tau}_g.
\end{aligned}
$$

Figure 4.2 shows the ground truth torques and the torques learned by DeLaN, i.e., the torque components and the super-imposed torque (Figure 4.2a-d). Even though DeLaN is only trained on the super-imposed torques, the result shows that it learns to disambiguate the inertial force $\mathbf{H}\ddot{\mathbf{q}}$, the Coriolis and Centripetal force $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$, and the gravitational force $\mathbf{g}(\mathbf{q})$. That can be seen as the respective curves overlap closely with the "true" torques. Hence, DeLaN is capable of

[2] The source code for PyBullet is available at `https://github.com/bulletphysics/bullet3`

[3] The framework is available at `https://pytorch.org/`

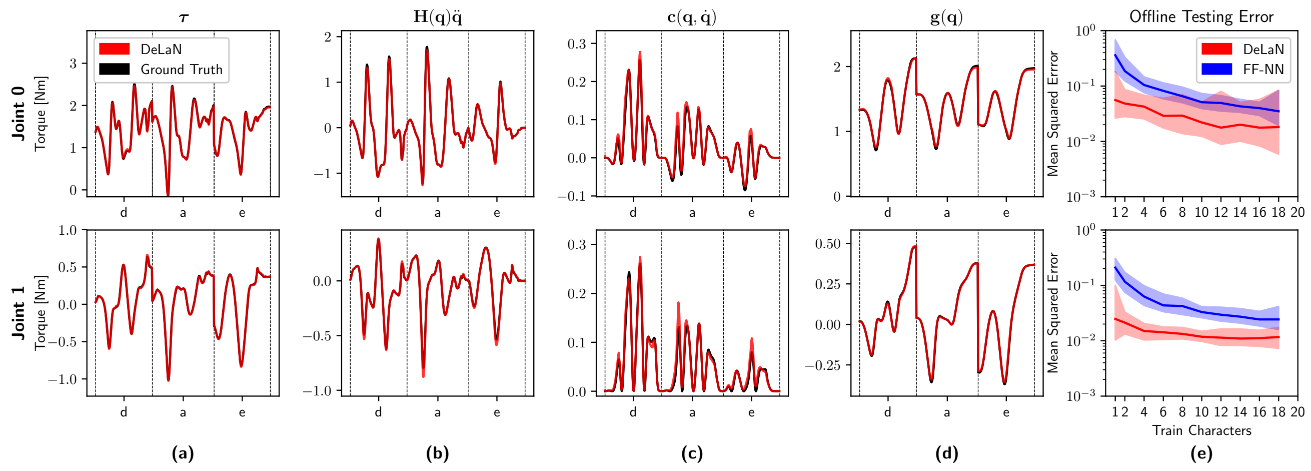


Figure 4.2.: Comparison between true and learned inverse dynamics model. The torque τ required to generate the characters 'a', 'd' and 'e' is plotted in black. Using six different characters as samples, Deep Lagrangian Network (DeLaN) was trained offline and learns the red trajectory. DeLaN can not only learn the desired torques but also disambiguate the individual torque components even though it was trained on the super-imposed torques (a). Using Equation (3.7) DeLaN can represent the inertial force $\mathbf{H}\ddot{\mathbf{q}}$ (b), the Coriolis and Centripetal forces $\mathbf{c}(\mathbf{q},\dot{\mathbf{q}})$ (c), and the gravitational force $\mathbf{g}(\mathbf{q})$ (d). All components match the ground truth data closely. (e) The offline MSE of the feed-forward neural network (FF-NN) and DeLaN for each joint over varying training set sizes. It highlight DeLaN's better extrapolation capabilities compared to the FF-NN.

learning the underlying physical model defined by the physics prior. Furthermore, the proposed network topology can be trained with standard end-to-end optimization on only the super-imposed torques.

Figure 4.2e shows the offline MSE on the test set averaged over multiple seeds for the FF-NN and DeLaN w.r.t. to different training set sizes. The different training set sizes correspond to the combination of n random characters, i.e., a training set size of 1 corresponds to training the model on a single character and evaluating the performance on the remaining 19 characters. DeLaN clearly obtains a lower test MSE compared to the FF-NN. Especially the difference in performance increases when the training set is reduced. This increasing difference on the test MSE highlights DeLaN's reduced sample complexity and the good extrapolation to unseen samples.

4.2.2 Online Learning Performance

The second experiment on the 2-dof robot is an online scenario as described in Section 4.1. First, a set of training trajectories is executed and online learning is performed. The characters used for training are randomly chosen and not part of the testing set. Furthermore, the training set is iterated multiple times to give the models enough time and samples to train. Figure 4.3 compares the accumulated tracking error per testing character of DeLaN and the FF-NN for four different training set sizes ($n = 1, 6, 8, 10$). It can be seen that DeLaN outperforms the FF-NN on every character and that DeLaN requires less training samples as the error for $n = 6$ is similar to the error for $n = 10$. Figure 4.4a shows the testing error averaged over all test characters. It highlights the better performance of DeLaN compared to the FF-NN. Furthermore, it shows that DeLaN extrapolates better to unknown samples as it requires less training characters than the FF-NN.

On the other hand, Figure 4.8 shows the qualitative comparison of the control performance. It depicts the resulting trajectories of test characters for different training set sizes, over multiple runs. It can be seen that DeLaN performs well on all characters even if only a small amount of characters was used for training. In contrast, the FF-NN performs much worse. Some character trajectories, like 'b', 'o', and 'q', are not executed properly even if twelve characters are used for training. While the analytical model performs perfect (as it resembles the underlying simulation) it can be seen that the PD-controller struggles with gravity and the high speed of the trajectory execution.

Overall the experiment shows that DeLaN has a lower sample complexity and better extrapolation to unseen trajectories compared to the FF-NN even when trained online. Moreover, the obtained tracking error is comparable to the analytical model which, in this case, contains the simulation parameters and, thus, is optimal. In contrast, the FF-NN shows significant deviation from the desired trajectories when trained on less than 10 random characters.

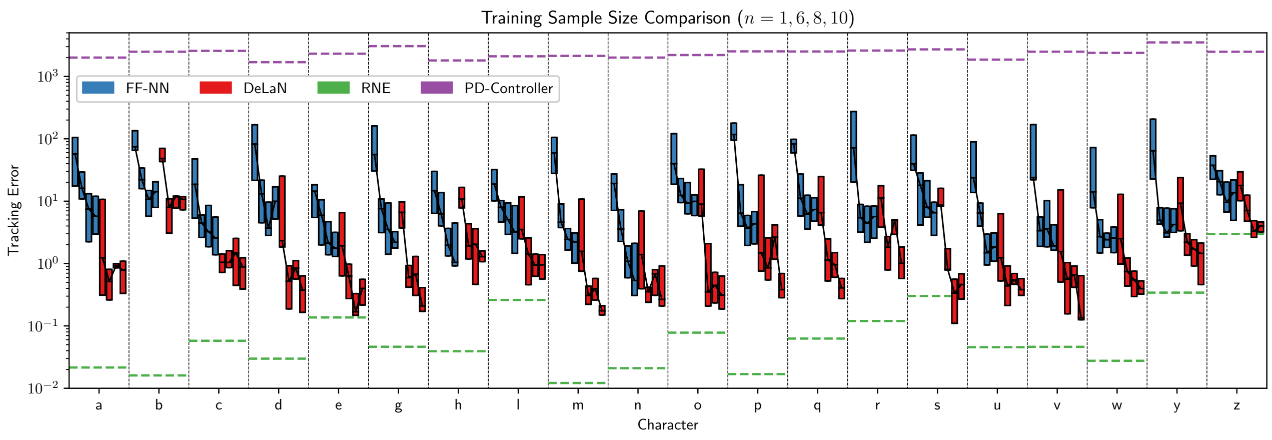


Figure 4.3.: The performance of Deep Lagrangian Network (DeLaN) and the feed-forward neural network (FF-NN) for each character. The 4 columns of the boxplots correspond to different numbers of training characters, i.e., $n = 1$, 6, 8, 10. Errors are accumulated over multiple runs as boxplots, i.e., the box spans from the first to the third quartile and the black line resembles the median. It can be seen that DeLaN outperforms the FF-NN on every character and requires less samples for a good overall performance. Both model's errors are one to two orders of magnitude smaller than those of the PD-controller while DeLaN is close to the analytical model (RNE) that is used in the simulation and, thus, is the lower bound.

4.2.3 Extrapolation to High Velocities

The third experiment on the 2-dof robot was performed on the cosine dataset. Here, the models are only trained online on two cosine trajectories with a velocity scale of 1x. The goal is to assess the capabilities of the model types to extrapolate w.r.t. velocities and accelerations. Thus, the learned models are tested on the same trajectories with multiple velocity scales.

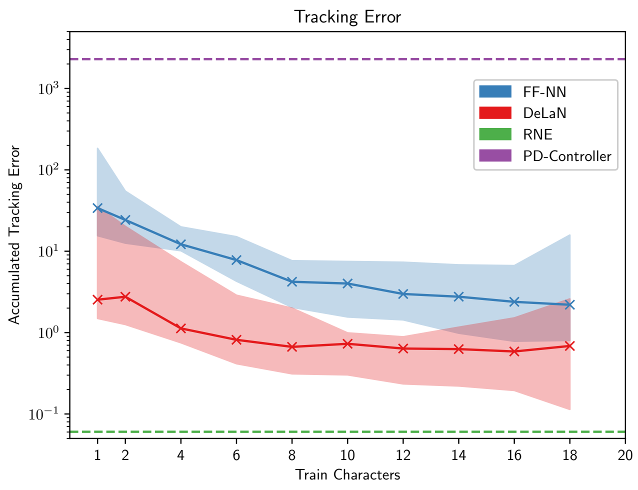
Figure 4.5a and b compare the performance of DeLaN and the FF-NN (the gray areas represent the test data where velocities are increased). It can be seen that DeLaN and the FF-NN perform comparable on the training trajectories. However, when the velocities are increased the performance of the FF-NN deteriorates because the new trajectories are no longer within the vicinity of the training data. This is due to the input domain of the FF-NN being defined as $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. Therefore, it cannot extrapolate to the testing data well. In contrast, the domain of the networks $\hat{\mathbf{L}}$ and $\hat{\mathbf{g}}$ composing DeLaN only consists of $\mathbf{q}$, and, thus, are not impacted by velocities or accelerations. This enables DeLaN, within limits, to extrapolate to the test trajectories with higher velocities. The increase in tracking error can be explained by the structure of $\hat{f}^{-1}$, where even small model errors scale quadratic with velocities. However, the obtained tracking error on the testing trajectories remains significantly lower compared to the FF-NN. It should also be noted that the performance of DeLaN for higher velocities is comparable to the analytical solution. This further confirms that the model learned by DeLaN is similar to the "true" physical model. All in all, this experiment highlights DeLaN's advantage over classical model learning approaches as it profits from the incorporated prior knowledge.

4.3 Physical Robot Experiments

In order to get results for a robot that is subject to true physics, experiments are conducted on the real robot Barett WAM (see Figure 4.4b). It features direct cable drives that are able to produce high torques generating fast and dexterous movements but yielding complex dynamics, which cannot be modelled using rigid-body dynamics. Therefore, the Barrett WAM is ideal for testing the applicability of model learning and analytical models[4] on complex dynamics.

Three experiments are performed on the WAM. First, I focus on the cosine trajectories as these trajectories produce dynamic movements. Note that the characters dataset cannot be used as the trajectories are mainly dominated by the gravitational forces. This experiment aims at confirming the results of the cosine experiment on the simulated 2-dof robot. The performance is evaluated in a simulation using SL[68] and on the real robot. Second, I try to learn the model analytical model from samples collected from the real Barrett WAM. This experiment targets the applicability of my physics prior to a mechanical system where its assumptions do not hold. Third, following the results of the first

[4] The analytical model of the Barrett WAM is obtained using a publicly available URDF [67]



(a) Accumulated tracking error

(b) The Barrett WAM

Figure 4.4.: (a) The median performance of Deep Lagrangian Network (DeLaN), the feed-forward neural network (FF-NN), the PD-controller and the analytical baseline (RNE) averaged over multiple seeds. The shaded areas mark the 5th to 95th percentile. It highlights the fact that DeLaN outperforms the FF-NN in every case which means that it extrapolates better to unknown samples.
(b) The Barrett WAM at the IAS lab in Darmstadt used for the real robot experiments.

experiment I try to evaluate what the reasons behind limitations of DeLaN are. The test is conducted offline on samples collected on the real robot.

4.3.1 Learning on a Complex Robot

The first experiment is similar to the cosine experiment on the 2-dof robot. Training is performed online with a velocity scale of $1\times$ which is increased during validation. Figures 4.5c and d show the tracking error on the cosine trajectories using the real (triangles) and the simulated (squares) Barrett WAM. It is important to note, that the simulation is only based on rigid-body dynamics and does not including the direct cables drives. Moreover, the simulation parameters are inconsistent with the parameters of the analytical model. Therefore, the analytical model is not optimal.

On the training trajectories executed on the physical system the FF-NN performs better compared to DeLaN and the analytical model. DeLaN achieves slightly better tracking error than the analytical model, which uses the same rigid-body assumptions as DeLaN. That shows that DeLaN can learn a dynamics model of the WAM but is limited by the model assumptions of Lagrangian Mechanics. These assumptions cannot represent the dynamics of the cable drives. On the other hand, it highlights that DeLaN has the potential to replace analytical models in the future. When looking at the results of the simulation, DeLaN and the FF-NN perform comparable but significantly better than the analytical model. These results highlight that DeLaN can learn an accurate model of a complex robot, as long as the underlying assumptions of the physics prior hold. This means that DeLaN can learn a model within the model class of the physics prior but also inherits the limitations of the physics prior. The experiment also shows that the FF-NN can locally learn correlations of the torques w.r.t. $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ while such cannot be represented by the network topology of DeLaN. In the definition of the pysics prior such correlation should not exist.

When increasing the velocity scale of the trajectories the tracking error of the FF-NN deteriorates much faster compared to DeLaN. That shows that the FF-NN cannot extrapolate to the new velocities neither in the simulation nor on the real robot. This may be due to the FF-NN overfitting the training data. In contrast, DeLaN can extrapolate to the higher velocities and maintains a good tracking error. Even further, it obtains a better tracking error compared the analytical model on all velocity scales in the simulation. This highlights the improved extrapolation of DeLaN compared to other model learning approaches. The tracking error of the analytical model remains constant and demonstrates the guaranteed extrapolation of analytical models.

In conclusion, this experiment has shown that the physics prior of DeLaN has the drawback of limiting the modeling

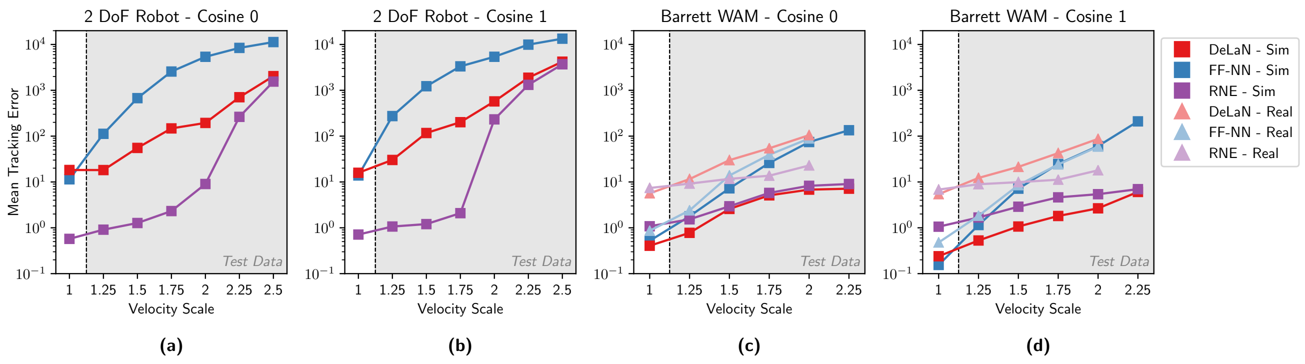


Figure 4.5.: The tracking error of the cosine trajectories for the simulated 2-dof robot (a and b), the simulated and the physical Barrett WAM (c and d). The feed-forward neural network (FF-NN) and Deep Lagrangian Network (DeLaN) are trained only on the trajectories at a velocity scale of $1\times$. Afterwards the models are tested on the same trajectories with increased velocities to evaluate the extrapolation to new velocities. (a) and (b) The analytical solution performs best which can be explained by the fact that it is the same model that is used for simulation. While the FF-NN and DeLaN perform on a similar level for velocity $1\times$ the first's performance deteriorates fast while DeLaN performs similar to the analytical model. (c) and (d) The results of the simulated WAM (squares) are similar to the results of the 2-dof robot. However, on the real WAM (triangles) the FF-NN outperforms DeLaN clearly. This originates in DeLaN's limited modeling power. On the other hand, DeLaN maintains a better extrapolation to higher velocities where the FF-NN fails.

power of the network. The FF-NN can learn any function that depends on the full joint state. However, DeLaN has the advantage of better extrapolation and a comparable performance to the analytical model. With some adaptations, DeLaN might be able to outperform the FF-NN while maintaining the advantage of extrapolation.

4.3.2 Learning the Analytical Model from Real Data

The second experiment evaluates whether DeLaN is able to learn the analytical model from real data, i.e., if the physics prior can be trained using end-to-end training. Similar to the first 2-dof robot experiment in Section 4.2.1 (see also Figure 4.2) a model is trained offline and then compared with the analytical model w.r.t. the individual parts of the torque. The data was collected on an online run on the cosine dataset on the real robot. The model is trained on a subset of the samples and evaluated on the rest of the samples.

Figure 4.6 shows the torque parts and the super-imposed torque on which the model was trained. The ground truth is plotted in black while DeLaN is coloured red and the analytical model is colored purple. It can be seen that the learned model is close to the analytical model while also following the real dynamics. This highlights the fact that the assumption of the physics prior enforces a physically plausible model. Furthermore, it shows that DeLaN can train such a model of a real mechanical system only by the super-imposed torque in an end-to-end fashion. On the other hand, it can be seen that DeLaN is not able to completely model the real torque which is due to limitations brought by the physics prior. This will be further discussed in the next section.

4.3.3 Evaluating the Limitations of DeLaN

In this final offline experiment I try to shed some light on the limitations introduced by the physics prior of DeLaN. Using data recorded on the real Barrett WAM, a model is trained. The MSE is then calculated for each sample and plotted over each element of the respective joint state. The goal is to find a correlation between some inputs and the error to assess possible additions to the physics prior that would increase the model performance without decreasing the physical plausability.

Figure 4.7 visualizes the median of the mean tracking error over absolute joint-wise velocities. Therefore, all samples are added to bins and then the median is calculated for each bin. Similarly, the shaded areas depict the 5-th to 95-th percentile of the error. The green dashed line shows the percentage of samples that are contained in each bin. It can be seen that an increasing velocity correlates with an increasing error. This hints that friction may be a missing part of the PDE that limits the current modeling power of DeLaN given the similarity to the Stribeck curve for viscous friction

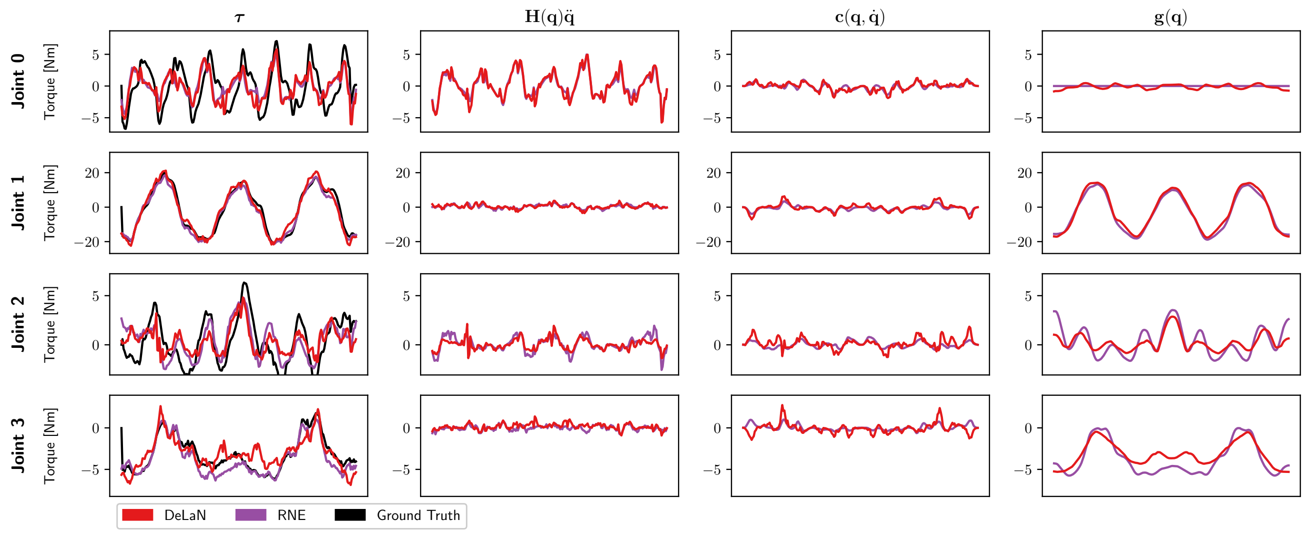


Figure 4.6.: Torques required to perform a cosine trajectory on the real Barrett WAM. Deep Lagrangian Network (DeLaN) (red) is trained offline on collected samples of the super-imposed torque. It is split into its components and compared to the analytical model (RNE) (purple). The true torque is plotted in black. It shows that DeLaN is able to model a real physical system while also maintaining the plausability of the physics prior.

that is also found in robotics [69]. On the other hand, the plots contain no sign of stiction, i.e., a higher MSE when the velocities are close to 0. Therefore, it may be another physical effect that is missing in my physics prior and this topic remains future work (see Section 5.3).

A full visualization of the error over all inputs can be found in the appendix, Figure A.1.

4.4 Seeding and Initialization Issues

During the experiments multiple random seeds were used for initialization and train / test set selection. For some seeds it appeared as if DeLaN did not converge to the desired model. It rather tried to approximate the mapping of the first joint using the gravitational term only. Further investigation into the problem revealed an issue with the actions taken in Section 3.2.1. In order to make $\mathbf{H}$ symmetrical it is expressed by a lower triangular matrix $\mathbf{L}$. Moreover, the diagonal entries of $\mathbf{L}$ are enforced to be positive by applying a non-negative activation.

Given how $\mathbf{H}$ is modeled a case can arise were three conditions are met. (1) all training samples are in vicinity of each other, (2) ReLU is used as activation function, and (3) the initial weights are in such a way that the activation of the first neuron, i.e., the first diagonal entry of $\mathbf{L}$, is 0. In that case the first row and column of $\mathbf{H}$ become $\mathbf{0}$ which leads to the output of the network for the first joint being 0 and, thus, no gradient will be passed through that node. After longer periods of training it is possible for the activation to become positive when the earlier network structure changes due to its shared nature between $\mathbf{l}$ and $\mathbf{g}$.

I propose two ways to mitigate this problem. First, using an non-linear function that does not output 0 for negative activations, like Softplus or the absolute, should not lead to the stagnation in training. Second, the initialization of the biases could be changed to force the first neuron to be active more often, i.e., the biases would be initialized with a higher order of magnitude.

A second, similar issue that occurred during a WAM experiment is that DeLaN may learn only a gravitational force for a joint and keeps its respective row and column of $\mathbf{H}$ at $\mathbf{0}$. This can happen if the joint movement is not influenced by gravity and the overall accelerations of that joint are low, i.e., there is only a small separate excitation w.r.t. $\mathbf{q}$ and $\ddot{\mathbf{q}}$. To mitigate this problem the training process has to be adjusted. One way is to initialize the gravity head of the network with lower values to have a higher gradient running through $\mathbf{l}_d$ and $\mathbf{l}_o$. This behaviour may also indicate a model architecture that is too complex, i.e., the model is overfitting. Decreasing the width and/or depth of the network can solve this issue.

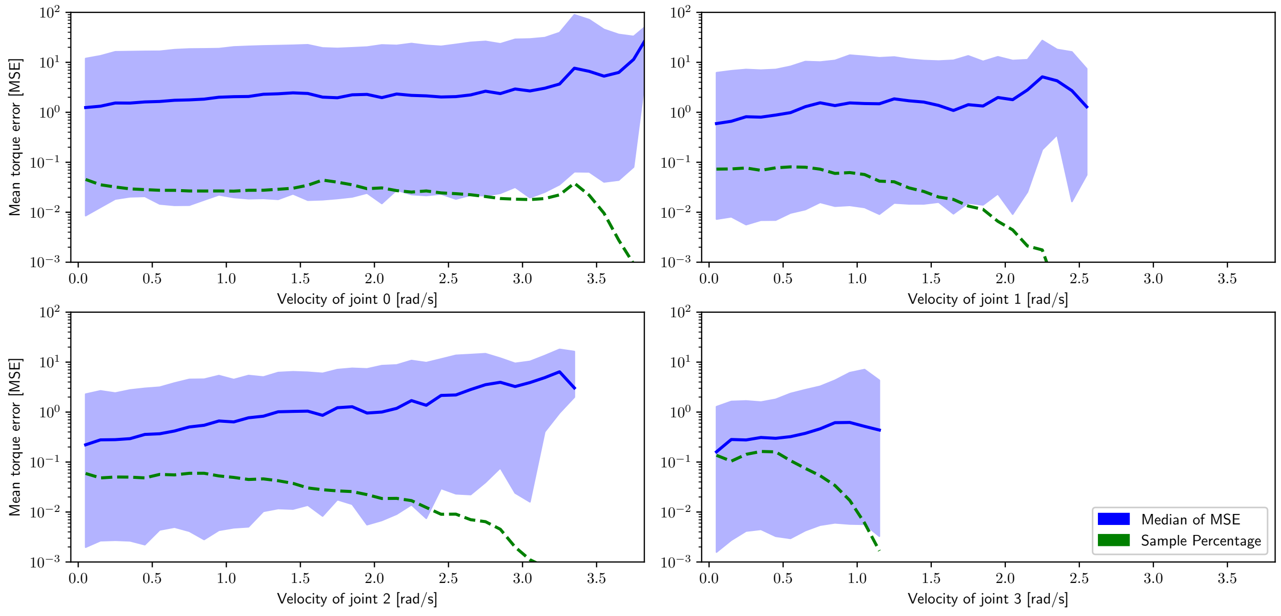


Figure 4.7.: Plots showing the median of the mean tracking error (MSE) (blue) obtained from training Deep Lagrangian Network (DeLaN) offline on data of the real robot Barrett WAM. All samples are added to bins of size 0.1 and all statistical values are calculated on those bins. The shaded area resembles the 5-th to 95-th percentile of the MSE. The dashed green lines depict the percentage of samples that are in each respective bin. The plots give insights in the correlation between individual velocities and the remaining error that cannot be modeled by DeLaN due to its physics prior. It can be seen that higher velocities result in a higher prediction error indicating that the current model assumption (Lagrange-Euler PDE) lacks a term representing friction.

4.5 Summary of Experiments and Results

In this chapter I conducted several experiments on a simulated 2-dof robot as well as on a simulated and a real Barrett WAM. I want to highlight two major insights from those experiments.

First, DeLaN is able to perform on a similar level to an analytical model of the robots with the advantage of good extrapolation to unseen samples. This is a trait not shared by the FF-NN. It also shows that DeLaN may be able to replace analytical models in the future as a model is easier to obtain but has similar advantages.

Second, DeLaN inherits the limitations of the physics prior. This means that DeLaN's performance gets worse when the model assumption is violated. One experiment revealed that one potential limitation may be friction as it is not modeled within the Lagrangian-Euler PDE. Adding additional terms to the current PDE could mitigate limitations and bring DeLaN closer to the modeling power of the FF-NN, but remains future work (see Section 5.3).

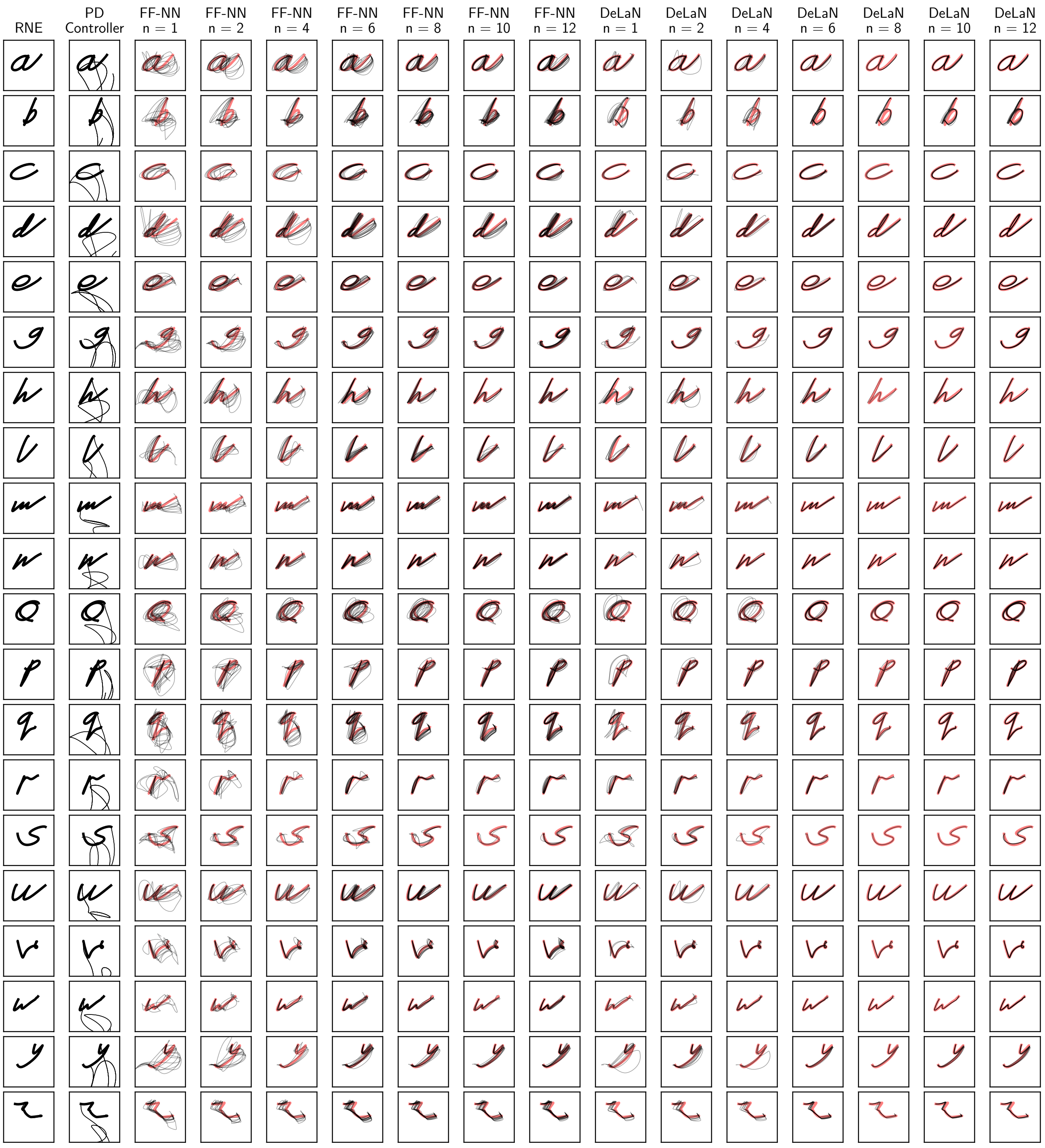


Figure 4.8.: The qualitative performance for the analytical baseline (RNE), the PD-controller, the feed-forward neural network (FF-NN) and Deep Lagrangian Network (DeLaN) for different numbers of random training characters. The desired trajectories are shown in black the trajectories resulting from the models in red. It can be seen that DeLaN achieves a good tracking performance when training on six to eight characters while the FF-NN extrapolates worse to unseen trajectories as it has a worse performance even if trained on twelve characters.

5 Discussion

In this chapter I summarize the most important insights obtained during the work on my thesis. I state advantages and disadvantages of DeLaN's architecture compared to a similar FF-NN in Section 5.1. This is followed by issues discovered regarding the convergence of the network optimization in Section 5.2 and the physical plausibility and limitations originating in the physics prior in Section 5.3. Those three conclude the major insights gained during my work. Section 5.4 and Section 5.5 describe future experiments that I would conduct. The first proposes to compare DeLaN to other machine learning algorithms besides the FF-NN. The second explains how the experiment setup could be changed to gain additional characteristics of DeLaN when applied to online learning. Section 5.6 highlights current fields of research where DeLaN could be used beneficially. At last, Section 5.7 suggests to derive and evaluate the forward model from a learned inverse model.

5.1 Architectural Advantages

In the experiments, DeLaN has proven to have the upper hand compared to the FF-NN when it comes to the network architecture. It especially profits from a smaller input space. As the training process is only reliant on the joint positions but not the velocities and accelerations, DeLaN has built-in means to tackle the problem of persistent excitation, i.e., it requires less samples overall to cover a greater variety of samples. While the extrapolation to unseen samples shown by DeLaN has not yet reached the same level as that of an analytical model it performs arguably better than a standard FF-NN.

Furthermore, early tests have shown that DeLaN is more robust to sensor noise which can be useful in a real environment were noise cannot always be avoided. Though, this has to be further studied to get an expressive quantitative evaluation.

Another advantage of DeLaN or any FF-NN in general is the natural applicability in online learning. As their training process is incrementally, i.e., optimization is repeatedly performed over mini-batches of the train samples, all FF-NNs have built-in ways to deal with adding new train data on-the-fly. Other machine learning approaches, like Linear Regression, Support Vector Regression, or Gaussian Process Regression require the full batch of training data instantly. Hence, they have to be retrained when new samples are added. Although, specialized versions of the aforementioned algorithms exist that tackle this issue by using locally weighted models or other adaptations to make online learning possible [12, 42, 14].

5.2 Network Convergence

As highlighted in Section 4.4, with its given structure DeLaN can be subject to convergence issues. This means that in some cases the training process gets stuck on a poor local optimum depending on the network initialization. I proposed two possible means to mitigate this issue, although they do not solve the problem entirely. Therefore, different initialization methods have to be tested when applying DeLaN to a new task.

In the future this convergence problem should be further investigated. One aspects to look at would be the network structure of DeLaN. Changes to the structure of the network head that outputs $\mathbf{l}_d$ are the most promising approach. An option might be to find other ways than using a ReLU to enforce the non-negativity of the entries. The problem of ReLU is that it outputs 0 for all negative values which results in a gradient of 0. Hence, the backward-pass ignores all parameters for that specific neuron and it is not optimized. This problem can be tackled by applying less strict non-linearities, e.g., the absolute or Softplus.

5.3 Physical Plausibility and Limitations

One of the issues stated at the beginning of this thesis is that classical model learning approaches lack the physical plausibility of analytical models derived from physical formalisms. This may be one of the main reasons why model learning is not yet state-of-the-art in robotics. As reliability is important, missing transparency of the models is an exclusion criterion.

However, due to the incorporation of a physics-prior, DeLaN adds a level of transparency to the learned model that a standard machine learning model cannot provide. The experiments have also shown that the model obtained through end-to-end training is similar to the analytical model which is based on the same physical assumption. This makes DeLaN a feasible alternative to the tedious process of measuring the mechanical system while maintaining the various applications of the matrices that form the Lagrange-Euler PDE.

On the backside, the physics prior limits the ability of the network to fit any arbitrary function. In Section 4.3.2 I described that behaviour where the model prior is learned in a similar fashion to the analytical model but some of the underlying physics could not be expressed. I tried to gain some insight into what is missing in the formulation of the prior in Section 4.3.3. It could be shown that the error correlates with the velocity which indicates that friction may be one of the missing terms of the Lagrange-Euler PDE.

Future work should focus on how to mitigate this limitation and how to make DeLaN more flexible while preserving physical plausibility. If DeLaN can perform on a similar level to the FF-NN while keeping its advantages over classical model learning it could be applied to tasks where reliability in control is important, e.g., in industrial robotics.

Adding a simple FF-NN to the equation that learns the torque that is not covered by DeLaN should produce good results while maintaining the level of physical plausibility. The network should be able to model the full correlation between torque and all inputs, i.e., it resembles a term $\nu(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \rightarrow \boldsymbol{\tau}$. Though, it has to be evaluated how to train such a network as classical end-to-end training might focus the optimization solely on the newly added FF-NN. This is similar to the issue discussed in Section 4.4. One possible solution would be to train the DeLaN first and introducing the non-linear term later in the training process to ensure it does only learn the missing correlation.

5.4 Additional Baselines

In my thesis I solely compare DeLaN only to a FF-NN and the analytical model. I chose the FF-NN because of its architectural similarity to DeLaN which makes the results comparable while keeping other experiment parameters fixed, e.g., the train time of both is on an even level. In order to get a broader comparison to the many machine learning models available future work could focus on reevaluating the experiments shown on additional model learning techniques.

I would suggest to evaluate Gaussian Process Regression (GPR) [40, 41] and Support Vector Regression (SVR) [43]. Both are frequently applied in machine learning and have been used as baselines before [12]. Hence, they also allow comparability of DeLaN to other approaches that have been evaluated against those two.

While the first has achieved great results in past experiments it requires longer training times due to a complexity of $O(n^3)$ [70]. Its applicability to my online learning system has to be tested. Moreover, it is a statistics-based machine learning approach and, thus, has high uncertainties in unknown regions of the input space, i.e., it should extrapolate worse than DeLaN. There are also approaches that decrease the training time of GPR, e.g., Local Gaussian Process Regression (LGP) [12]. SVR, on the other hand, should train a bit faster than GPR, given a slightly lower complexity [71], but is not expected to be faster than the DeLaN once a high amount of train samples is added. There are also existent approaches were SVR is used in online learning using local models [42]. Another algorithm that could be tested and has been used for inverse dynamics in the past is Locally Weighted Projection Regression (LWPR) [14] or other locally weighted regression methods [9]. They all feature a low time complexion w.r.t. the number of train samples, e.g., LWPR has time complexity $O(n)$.

I would conduct all proposed experiments on the simulated robots and the real robot using the mentioned algorithms and their online versions. Thus, an upper limit for those algorithms is obtained, i.e., their offline training performance, as well as the performance of their online counterparts. From there, the approaches can be compared to DeLaN to determine whether it outperforms current online learning approaches and to compare future adaptations made to DeLaN with state-of-the-art regression methods. The ultimate goal for DeLaN is to outperform offline learning algorithms with online training times.

5.5 Evaluation with Varied Experimental Setup

Another aspect that should be evaluated is the performance of the models on different datasets or experimental setups. Especially on the real WAM it would be interesting to know which model is suited best for which kind of task given that I only evaluated the real robot on a single dataset. For example, a FF-NN might be best for slow movements while DeLaN performs better if larger portions of the task space have to be covered.

In general, online learning is steered by the samples given to the training process. The better extrapolation w.r.t. unseen samples of DeLaN compared to the FF-NN could be even more beneficial the more complex a task gets. Therefore, I suggest to evaluate the extrapolation of the model learning algorithms w.r.t. task space regions. In this thesis I tested extrapolation by increasing the velocity and acceleration of a known trajectory. In contrast, the models could be trained on trajectories that cover only a part of the task space and could then be evaluated on trajectories in other regions of the task space. If DeLaN would maintain results usable in a real scenario due to its physics prior it could be used as a substitute for an analytical model. Although, DeLaN may fail in such a scenario due to $\hat{\mathbf{l}}$ and $\hat{\mathbf{g}}$ depending on $\mathbf{q}$.

Furthermore, noise could be added to the sensor measurements. In every real scenario noise is present, and especially in remote controlled environments this might be more of an issue than in the use case of this thesis. While analytical models are robust to noise this is not ensured for model learning approaches. Given the incorporated PDE of DeLaN it should handle noise better than the FF-NN or other model learning approaches. Hence, I would conduct experiments with a varying level of noise. Every parameter of the experiment stays fixed, i.e., dataset, number of training samples, model hyperparameters, etc. Then from the same random initialization (using a fixed seed for all pseudo-random numbers) the models are trained online with a different amount of Gaussian noise added to the sensor measurements. The tracking error would then be compared to measure the models' robustness.

Another aspect that could be tested is long-term online learning. Imagine a robot that autonomously performs a task while its dynamics model is learned online. As it is previously unknown which movements the robot has to do there is no prior knowledge of when the learning process could be stopped. This may lead to overfitting and, hence, the model becoming worse from some point in time onward. While the experiments conducted indicate that DeLaN is less susceptible to overfitting than the FF-NN the impact of long-term training has yet to be evaluated. Given the outcome, there are multiple options to deal with overfitting. One that is already in place is regularization. Regularization is basically the addition of a term to the optimization objective function that penalizes high parameter values. I introduced it to DeLaN's optimization problem in Equation 3.10. A second means to deal with overfitting is early-stopping using a validation set [72]. Here, the learning process is ended once a given criterion is fulfilled, e.g., the validation error did not decrease for multiple epochs. It would require to split a small amount of samples from the training data for validation or to alter the training process, e.g., when using k-fold cross-validation. Though, early stopping may be harmful as it prevents adaptation to changing dynamics once the learning is stopped. This can be an issue in a scenario where the dynamics model changes at runtime, for example if a robot at a remote location gets damaged. A third, and maybe most promising, option is to include dropout layers [73] into DeLaN's network architecture. By randomly deactivating neurons in the training process, dropout layers effectively decrease the overfitting capabilities of a neural network while maintaining a high level of prediction accuracy.

The final experiment I would perform is to evaluate the adaptability of DeLaN to changes in the dynamics model. One example where this may occur is when a robot is grabbing a heavy object. An analytical model cannot compensate changing dynamics and it would be required to derive a model for each possible configuration of robot and grabbed object. DeLaN should be able to adapt to changing dynamics through continuous online learning. An experiment would be to start learning on a trajectory, e.g., the cosine trajectory, and after a few iterations adding a weight to the robots end-effector. The performance of all models could then be compared on two measurements. First, the resulting tracking error and, second, the time required for the error to decrease to a level similar to the error before the weight was added. One parameter that could be crucial for fast adaptation is the total amount of samples kept in the training set. On one hand, a larger size would require more time until all samples are replaced. On the other hand, a smaller training set may limit the overall power of DeLaN.

5.6 Exporting DeLaN to Other Fields of Research

In this section I want to showcase some current fields of research where DeLaN could be applied. Those mentioned do not cover all possible options but are rather a few interesting examples.

One currently heavily investigated field of research is model-based reinforcement learning (RL) [74, 75, 76, 77]. Model-based RL utilizes model learning to obtain required but unknown models, e.g., the state-transition model. Once a model is learned, model-free RL techniques could be applied [78]. Many RL approaches rely on the dynamics model of a system [75, 76, 77]. DeLaN could be used in such cases where a dynamics model is needed. Through the physics prior the model should be trained fast while adding physical plausibility. Furthermore, the prior would allow to control which model class is learned. Although it could be necessary to adapt the prior to model the dynamics function of the respective system, as explained earlier, DeLaN could easily be adapted by changing the incorporated PDE.

Furthermore, DeLaN can be applied in optimal control. The combination of machine learning and optimal control is an ongoing research effort [79]. DeLaN can be integrated as model into an optimal controller and by minimizing a given cost functional, i.e., the loss function used for training, optimality is obtained. Optimal control is also closely related to RL and, thus, DeLaN and optimal control could be combined using model-based RL [80].

A third possible future application of DeLaN is in compliant control. The goal of compliant control is to react to forces that the environment acts upon the system. Compliant models can be learned [81, 82, 83] and, as they are dependent on PDEs, this can be done using DeLaN. In cases where compliant control is desired which are often cases where a human interacts with a robot the reliability of the system's compliance is mandatory. Through the physics prior of DeLaN it can be assured that the learned model follows a given directive. Thus, DeLaN may be able to model a compliant controller while being trained in complete end-to-end fashion.

DeLaN could also be used in applications where any dynamics model is required as the prior could be adapted easily. One interesting application that is currently investigated would be the dynamics of microbiomes [84], another one would be fluid dynamics [85].

5.7 Deriving and Using the Forward Model

Another experiment, though more complex to realize, is to derive and use the forward dynamics model from DeLaN. As shown in Equation (3.9) the matrix $\mathbf{L}$ and vector $\mathbf{g}$ which are the output of DeLaN could be used to derive the forward dynamics model $\hat{f}$. The experiment would be to train DeLaN on an inverse dynamics task, as shown in this thesis. $\hat{f}$ could then be derived using the learned network and applied to a task like model-predictive control or optimal control. Here, DeLaN's performance could hint the usability of such a derived forward model and, if results are sufficient, this could be another major advantage of DeLaN making it more versatile.

6 Conclusion

The focus of this thesis was the application of deep learning as means of modeling inverse dynamics of a real system, i.e., a real robot. The contribution of this work is a twofold. First, the concept of incorporating a physics prior within a deep learning framework was proposed. All necessary mathematical equations were derived and the network architecture of a standard feed-forward neural network (FF-NN) was extended to facilitate encoding a partial differential equation (PDE) as part of the network. Second, experiments were conducted on this concept to determine its advantages and drawbacks as well as its real-time applicability and, thus, usability in robot control. These experiments were performed both in simulations and on a real, physical robot.

In particular, I first proposed Deep Lagrangian Network (DeLaN), a deep neural network upon which Lagrangian Mechanics is imposed. This specific network topology allows to learn the inverse dynamics of any mechanical system. Thereby it combines the advantages of classical model learning, the possibility to use end-to-end training from samples, with the physical plausibility of an analytical model. The necessary mathematical derivations to incorporate a PDE into a neural network were derived. From that a new feed-forward network layer, the Lagrangian layer, was proposed that extends the standard layer by passing the derivative of the layer w.r.t. the previous layer. In the case of inverse dynamics model learning DeLaN reduces the required input space to a third. This makes the resulting model more robust and reduces the amount of samples needed for a sufficient model. Furthermore, a system architecture that allows the asynchronous training and control of a mechanical system has been showcased. The architecture enables real-time control while facilitating online learning allowing to profit from its advantages, e.g., not needing to gather training samples previous to the execution of the task or the problem of persistent excitation.

The experiments have shown that DeLaN is able to learn the underlying physics from a super-imposed signal, i.e., it can recover the contribution of the inertial-, gravitational, Coriolis, and Centripetal forces from sensor data. In quantitative evaluations within a real-time control loop I assessed the tracking error on the task that should be performed by the robot. The evaluations showed that DeLaN can learn the system dynamics online while obtaining lower sample complexity and better generalization compared to a FF-NN. DeLaN also performs close to an analytical model, derived by the Recursive Newton Euler (RNE) algorithm, which highlights the possible advantages compared to classical machine learning approaches. Moreover, DeLaN can extrapolate to new trajectories as well as to increased velocities, where the performance of the FF-NN deteriorates due to overfitting to the training data. This is an issue that most classical model learning approaches are subject to and may be one of the main reasons analytical models are still the state-of-the-art in current robot applications.

However, experiments also discovered that, when applied to a real physical system with complex dynamics, the bounded representational power of the physics prior may be limiting. This can be explained by the fact that DeLaN can only approximate models that are based on the same physical assumption. Anything that is not modeled in that assumption cannot be learned. In case of the Lagrange-Euler PDE this is true for, e.g., friction. But, through this limited representational power the physical plausibility is enforced.

In future work the physics prior should be extended to represent a wider system class by introducing additional non-conservative forces within the Lagrangian. It then has to be evaluated whether the training process has to be adapted to keep the physical plausibility intact. Moreover, some additional experiments should be conducted on DeLaN. DeLaN should be compared to other machine learning algorithms in the future. This would facilitate ranking DeLaN among the wide variety of available model learning options. A broader comparison could also highlight scenarios where DeLaN outperforms existing approaches or where it should not be used. Another experiments worth doing would be the evaluation of DeLaN's robustness to noise and adaptability to changing dynamics. The latter is something that especially an analytical model is unable to describe and, thus, would be a major point for DeLaN. Furthermore, the impact of long-term online training has yet to be evaluated. This is important in order to use DeLaN in a prolonged scenario where online training and adaptability is required, e.g., in a scenario with a remote, autonomous robot.

DeLaN should also be applied to other fields of research. One prominent application where DeLaN could shine is model-based reinforcement learning. Reinforcement learning often requires dynamics models, but as they are usually unknown, they have to be learned. This is a case where DeLaN with its physics prior could be integrated relatively easy into existing approaches [75, 76, 77] adding benefits like physical plausibility, extrapolation to unknown states, and a more controlled training process. Additionally, DeLaN could be applied to various control scenarios, like optimal or compliant control. Combinations of either of the two with machine learning techniques is currently investigated [79, 81, 82, 83] and, hence, possible use cases for DeLaN could be found. Other fields of research that depend on any system dynamics could also make use of DeLaN when adapting the incorporated PDE. Two examples would be the dynamics of microbiomes [84] or fluid dynamics [85].

To conclude, Deep Lagrangian Network (DeLaN) is a novel means of modeling inverse dynamics of robots, or any mechanical system. Given the results of my experiments it could become a reasonable alternative to analytical models in applications like production lines or human-robot-interaction where a reliable model is necessary. This would save great amounts of money and time that currently have to be spent on engineering the analytical models.

Bibliography

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

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] G. Hinton, “Neural networks for machine learning,” *Coursera*, 2012.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[6] A. Albu-Schäffer, *Regelung von Robotern mit elastischen Gelenken am Beispiel der DLR-Leichtbauarme*. PhD thesis, Technische Universität München, 2002.

[7] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[8] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, “Learning inverse dynamics models in o (n) time with lstm networks,” in *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*, pp. 811–816, IEEE, 2017.

[9] S. Schaal, C. G. Atkeson, and S. Vijayakumar, “Scalable techniques from nonparametric statistics for real time robot learning,” *Applied Intelligence*, vol. 17, no. 1, pp. 49–60, 2002.

[10] J. D. Olden and D. A. Jackson, “Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks,” *Ecological Modelling*, vol. 154, no. 1, pp. 135 – 150, 2002.

[11] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding Neural Networks Through Deep Visualization,” *ArXiv e-prints*, 2015.

[12] D. Nguyen-Tuong, M. Seeger, and J. Peters, “Model learning with local gaussian process regression,” *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[13] D. Nguyen-Tuong and J. Peters, “Model learning for robot control: a survey,” *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[14] S. Vijayakumar, A. D’Souza, and S. Schaal, “LWPR: A scalable method for incremental online learning in high dimensions,” 2005.

[15] M. Lutter, C. Ritter, and J. Peters, “Deep lagrangian networks: Using physics as model prior for deep learning,” in *Submitted to International Conference on Learning Representations*, 2019. under review.

[16] D. Nguyen-Tuong and J. Peters, “Incremental online sparsification for model learning in real-time robot control,” *Neurocomputing*, vol. 74, no. 11, pp. 1859 – 1867, 2011.

[17] M. Kawato, “Feedback-error-learning neural network for supervised motor learning,” in *Advanced Neural Computers*, pp. 365 – 372, Amsterdam: North-Holland, 1990.

[18] J. Nakanishi and S. Schaal, “Feedback error learning and nonlinear adaptive control,” *Neural Networks*, vol. 17, no. 10, pp. 1453 – 1465, 2004.

[19] M. I. Jordan and D. E. Rumelhart, “Forward models: Supervised learning with a distal teacher,” *Cognitive Science*, vol. 16, no. 3, pp. 307 – 354, 1992.

[20] J. J. Craig, *Introduction to robotics: Mechanics and Control*, vol. 3. Pearson/Prentice Hall, Upper Saddle River, NJ, USA, 2005.

[21] P. A. Ioannou and J. Sun, *Robust adaptive control*, vol. 1. PTR Prentice-Hall Upper Saddle River, NJ, 1996.

[22] C. C. de Wit, B. Siciliano, and G. Bastin, *Theory of robot control*. Springer Science & Business Media, 2012.

[23] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer Science & Business Media, 2013.

[24] K. Zhou, J. C. Doyle, K. Glover, *et al.*, *Robust and optimal control*, vol. 40. Prentice hall New Jersey, 1996.

[25] G. Calafiore, M. Indri, and B. Bona, "Robot dynamic calibration: Optimal excitation trajectories and experimental parameter estimation," *Journal of Robotic Systems*, vol. 18, no. 2, pp. 55–68, 2001.

[26] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 3, pp. 49–59, 1987.

[27] P. K. Khosla and T. Kanade, "Parameter identification of robot dynamics," in *1985 24th IEEE Conference on Decision and Control*, pp. 1754–1760, 1985.

[28] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[30] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 315–323, PMLR, 2011.

[31] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," in *Advances in Neural Information Processing Systems 13*, pp. 472–478, MIT Press, 2001.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, p. 533, 1986.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[35] A. Griewank, "Who invented the reverse mode of differentiation?," *Documenta Mathematica, Extra Volume ISMP*, pp. 389–400, 2012.

[36] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall, 2nd ed., 1999.

[37] M. Haruno, D. M. Wolpert, and M. Kawato, "Mosaic model for sensorimotor learning and control," *Neural computation*, vol. 13, no. 10, pp. 2201–2220, 2001.

[38] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 44–54, 2010.

[39] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[40] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, "Gaussian process model based predictive control," in *American Control Conference, 2004. Proceedings of the 2004*, vol. 3, pp. 2214–2219, IEEE, 2004.

[41] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics.," in *ICRA*, pp. 2677–2682, 2010.

[42] Y. Choi, S.-Y. Cheong, and N. Schweighofer, "Local online support vector regression for learning control," in *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007.*, pp. 13–18, IEEE, 2007.

[43] J. P. Ferreira, M. Crisostomo, A. P. Coimbra, and B. Ribeiro, "Simulation control of a biped robot with support vector regression," in *Intelligent Signal Processing, 2007. WISP 2007.*, pp. 1–6, IEEE, 2007.

[44] M. Jansen, “Learning an accurate neural model of the dynamics of a typical industrial robot,” in *Int. Conf. on Artificial Neural Networks*, pp. 1257–1260, 1994.

[45] I. Lenz, R. A. Knepper, and A. Saxena, “Deepmpc: Learning deep latent features for model predictive control.,” in *Robotics: Science and Systems*, 2015.

[46] F. D. Ledezma and S. Haddadin, “First-order-principles-based constructive network topologies: An application to robot inverse dynamics,” in *17th International Conference on Humanoid Robotics (Humanoids). 2017 IEEE-RAS.*, pp. 438–445, IEEE, 2017.

[47] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, “Graph networks as learnable physics engines for inference and control,” *arXiv preprint arXiv:1806.01242*, 2018.

[48] C. G. Atkeson, C. H. An, and J. M. Hollerbach, “Estimation of inertial parameters of manipulator loads and links,” *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 101–119, 1986.

[49] B.-G. Hu, H.-B. Qu, Y. W., and S.-H. Yang, “A generalized-constraint neural network model: Associating partially known relationships for nonlinear regressions,” *Information Sciences*, vol. 179, no. 12, pp. 1929 – 1943, 2009.

[50] K. S. Narendra and A. M. Annaswamy, “Persistent excitation in adaptive systems,” *International Journal of Control*, vol. 45, no. 1, pp. 127–160, 1987.

[51] B. F. Hobbs and A. Hepenstal, “Is optimization optimistically biased?,” *Water Resources Research*, vol. 25, no. 2, pp. 152–160, 1989.

[52] W.-K. Mak, D. P. Morton, and R. K. Wood, “Monte carlo bounding techniques for determining solution quality in stochastic programs,” *Operations research letters*, vol. 24, no. 1-2, pp. 47–56, 1999.

[53] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[54] I. E. Lagaris, A. Likas, and D. G. Papageorgiou, “Neural-network methods for boundary value problems with irregular boundaries,” *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[55] M. Raissi and G. E. Karniadakis, “Hidden physics models: Machine learning of nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 357, pp. 125–141, 2018.

[56] J. Sirignano and K. Spiliopoulos, “DGM: A deep learning algorithm for solving partial differential equations,” *arXiv preprint arXiv:1708.07469*, 2017.

[57] Z. Long, Y. Lu, X. Ma, and B. Dong, “PDE-Net: Learning PDEs from data,” *arXiv preprint arXiv:1710.09668*, 2017.

[58] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations,” *arXiv preprint arXiv:1711.10561*, 2017.

[59] S. Sahoo, C. Lampert, and G. Martius, “Learning equations for extrapolation and control,” in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, pp. 4442–4450, PMLR, 2018.

[60] D. T. Greenwood, *Advanced dynamics*. Cambridge University Press, 2006.

[61] R. Featherstone, *Rigid Body Dynamics Algorithms*. 2007.

[62] B. Williams, M. Toussaint, and A. J. Storkey, “Modelling motion primitives and their timing in biologically executed movements,” in *Advances in Neural Information Processing Systems 20*, pp. 1609–1616, 2008.

[63] D. Dheeru and E. Karra Taniskidou, “UCI machine learning repository,” 2017. Available at `http://archive.ics.uci.edu/ml`.

[64] J. Y. Luh, M. W. Walker, and R. P. Paul, “On-line computational scheme for mechanical manipulators,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 102, no. 2, pp. 69–76, 1980.

[65] E. Coumans and Y. Bai, “PyBullet, a python module for physics simulation for games, robotics and machine learning.” `http://pybullet.org`, 2016–2018.

[66] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256, PMLR, 2010.

[67] JHU LCSR, "Barrett model containing the 7-dof urdf," 2018. Available at `https://github.com/jhu-lcsr/barrett_model`, last checked: 2018-09-18.

[68] S. Schaal, "The SL simulation and real-time control software package," tech. rep., Los Angeles, CA, 2009.

[69] A. C. Bittencourt and S. Gunnarsson, "Static friction in a robot joint — modeling and identification of load and temperature effects," *Journal of Dynamic Systems, Measurement, and Control*, vol. 134, no. 5, 2012.

[70] C. K. I. Williams and C. E. Rasmussen, "Gaussian processes for regression," in *Advances in Neural Information Processing Systems 8*, pp. 514–520, MIT Press, 1996.

[71] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007.

[72] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in Neural Information Processing Systems 13*, pp. 402–408, MIT Press, 2001.

[73] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[74] P. Parmas, C. E. Rasmussen, J. Peters, and K. Doya, "PIPPS: Flexible model-based policy search robust to the curse of chaos," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, pp. 4065–4074, PMLR, 2018.

[75] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving PILCO with bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, ICML*, 2016.

[76] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *CoRR*, vol. abs/1805.12114, 2018.

[77] S. Kamthe and M. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, vol. 84, pp. 1701–1710, PMLR, 2018.

[78] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[79] G. Lee, S. S. Srinivasa, and M. T. Mason, "GP-ILQG: Data-driven robust optimal control for uncertain nonlinear dynamical systems," *CoRR*, vol. abs/1705.05344, 2017.

[80] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2042–2062, 2018.

[81] C. Li, Z. Zhang, G. Xia, X. Xie, and Q. Zhu, "Efficient force control learning system for industrial robots based on variable impedance control," *Sensors*, vol. 18, no. 8, 2018.

[82] T. Petrič, A. Gams, L. Colasanto, A. J. Ijspeert, and A. Ude, "Accelerated sensorimotor learning of compliant movement primitives," *IEEE Transactions on Robotics*, pp. 1–7, 2018.

[83] J. Schreiter, P. Englert, D. Nguyen-Tuong, and M. Toussaint, "Sparse gaussian process regression for compliant, real-time robot control," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2586–2591, 2015.

[84] T. Gibson and G. Gerber, "Robust and scalable models of microbiome dynamics," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 1763–1772, PMLR, 2018.

[85] J. Morton, F. D. Witherden, A. Jameson, and M. J. Kochenderfer, "Deep dynamical modeling and control of unsteady fluid flows," *CoRR*, vol. abs/1805.07472, 2018.