Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

Physically constrained networks

Korbinian Abstreiter

Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

Physically constrained networks

Physikalisch bedingte Neuronale Netzwerke

Author:	Korbinian Abstreiter
Supervisor:	Prof. Dr.-Ing. Laura Leal-Taixe
Advisor:	M.Sc. Patrick Dendorfer
Submission date:	July 15th, 2019

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

_________________________ _______________________________________

Date Korbinian Abstreiter

Abstract

Due to the rise of deep learning models in recent years, they are nowadays applied to a wide range of different tasks. Previously, problems could only be solved by hand-crafted models, which had to be designed individually for each task. Such models are based on principles that scientists and mathematicians have discovered and experimentally validated over many years. However, it does not only require much more effort to create a hand-crafted model, but there is also a high number of problems that cannot be solved analytically, including many examples of differential equations. In contrast, deep learning models can be applied whenever training data is available, even when theoretical knowledge is sparse. However, the biggest limitation of data-driven models is the limited amount of available data for real world applications. As a consequence, their predictions deviate from the correct values and often do not align with known scientific principles. This raises the question if we purely have to rely on data to learn underlying mathematical theories which we know of and assume to be true.

In this thesis, we examine and experiment with different approaches to incorporate knowledge of physical constraints into deep learning models. In particular, we analyse the Penalty Method and Augmented Lagrangian Method. In addition, we inspect the results when projecting predictions according to physical constraints during training. The benefits of these approaches are validated on the task of predicting a rotation in two and three dimensions with the constraints on the determinant of the rotation matrix and the preservation of the norm. We confirm that the methods lead to significantly higher overall performance even for small training datasets. We further show that incorporating the information about physical constraints leads to predictions with smaller constraint violations. This is particularly true in regions where no or only few training data is available.

Contents

1 Introduction

In recent years, deep learning models have proven to perform well in a wide range of real world applications. For example, neural networks have successfully been applied to the task of image classification [7], text translation [4, 10] and autonomous driving [6, 1]. The main reason for their rise in popularity is their ability to learn the underlying system structure and scientific principles solely from training data.

Previous to the breakthrough of deep learning, problems were tried to be solved using hand-crafted models. The latter were designed in a way to align with physical principles, such as gravity, the conservation of energy, or Newton's laws. Therefore, scientists desired to formulate mathematical theories and principles, often including empirically estimated parameters such as the gravitational constant or the speed of light. The approach of creating hand-crafted models to align with scientific principles leads to high precision and avoids the problem of generalisation, since the theories are assumed to be true across the whole domain space. Nevertheless, manually building models requires intense effort and domain knowledge, and their performance may suffer from flawed parameter estimates or theories. In particular, they can only be used when an analytical method to obtain a solution is known.

In contrast, deep learning models do not rely on understanding the complex underlying system structure, but instead are able to learn the behaviour solely from previously observed data. For this reason, data-driven models can be used even when no scientific theories are known. Furthermore, they can in theory be applied universally to any problem, since they can approximate any continuous function [5]. However, utilising deep learning models comes at the high cost of requiring large amounts of training data for them to create realistic predictions aligning with physical principles. In addition, the data has to cover the whole domain space, which is often not the case. A lack of data in one region will already lead to model predictions which violate scientific constraints, since the deep learning model does not specifically learn the physical rules and can therefore not generalise well.

In this thesis, we combine the advantages of scientific knowledge with the power of deep learning models. We aim to show that theoretically constrained neural networks achieve higher performance and that their predictions are physically feasible. Further, the additional information about physical constraints reduces the required amount of training data. In order to show these improvements, we train a deep learning model to output predictions that satisfy all physical constraints. To do so, we apply methods for constrained optimisation. In particular, we analyse the results when using the Penalty Method and the Augmented Lagrangian Method. We further experiment with Physical Projection, a method which projects the predictions to the closest feasible point. In order to examine the true effects of these methods, we observe both the performance of the model on a test dataset and the physical feasibility of the predictions, and compare these statistics to the ones of a baseline model which is trained without the information about physical constraints.

By applying the methods mentioned above to the problem of learning a rotation, we achieve significant performance improvements when utilising the Penalty Method and the Augmented Lagrangian Method. This is particularly true for small amounts of training data. We further show that the predictions of the resulting models satisfy physical constraints more accurately than the baseline model.

Furthermore, we need to mention that the methods examined in this thesis are not limited to physically constrained problems. Instead, they can be applied to any problem where rules or constraints are known to be satisfied by the ground truth. For example, in autonomous driving, one can explicitly inform the model to only drive on streets.

We begin the thesis by giving a broad overview of optimisation problems and important solving approaches in section 2.1. This is followed by theoretical background information about constrained optimisation problems in section 2.2. We continue in section 2.3 by explaining the solving methods we apply and give information about their theoretical foundation and limitations. In section 3, we explain the framework of our experiments and the methods in detail. We propose several model architectures to solve the problem of learning a rotation, inform about technical details, and address the methods used to evaluate the trained models. The results of our experiments are visualised and discussed in section 4. Finally, we discuss the key findings of this thesis and give an outlook on what remains to be explored by future studies.

2 Theoretical Background

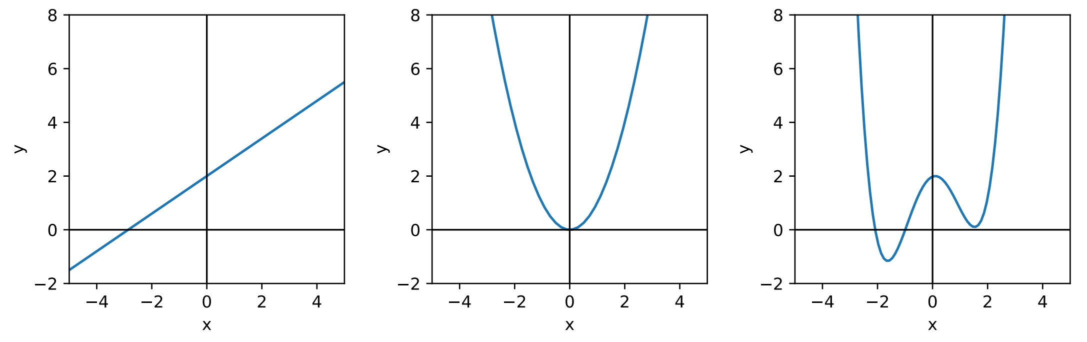
In this section, we begin with introducing the general problem of optimisation. This is followed by the formulation of constrained optimisation problems, for which we give a more detailed description of different solving approaches. In particular, we focus on the Penalty Method and the Augmented Lagrangian Method.

2.1 Optimisation problems

Optimisation problems are subject to mathematicians and scientists in finding the best solution for a given problem. Formally, an optimisation problem can be described as minimising an objective function f, that is,

$$\underset{x \in \mathbb{R}^n}{\operatorname{argmin}} f(x), \tag{1}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ is the smooth function to be minimised and $n \in \mathbb{N}$ is the input dimension. In general, optimisation problems can be divided into several categories depending on properties of the objective function f. In the following, we briefly describe the most important categories of unconstrained optimisation problems.



(a) $f(x) = 0.7x + 2$ (b) $f(x) = x^2$ (c) $f(x) = 0.4x^4 - 2(x - 0.1)^2 + 2$

Figure 1: Example for a strictly monotonic function (1a), a strictly convex function (1b) and a function with different local and global minima (1c).

Strictly monotonic functions

The first category of minimisation problems are the ones with a strictly increasing (resp. decreasing) objective function f. Formally, for $n = 1$, this is the case if $\forall a, b \in \mathbb{R} : a > b \Rightarrow f(a) < f(b)$ (resp. $f(a) > f(b)$). All linear functions of the form $f(x) = mx + t$ with $m \neq 0$ are strictly monotonic. An example for a linear and strictly monotonic function is shown in Figure 1a. However, these functions do not have any global minima in an open domain and the optimisation problem therefore has no solution. For bounded domain spaces, the global minimum is located on the boundary of the domain.

Strictly convex functions

In addition to strictly monotonic functions, we want to introduce optimisation problems with strictly convex objective functions. Formally, a function is strictly convex if it satisfies the following condition:

$$\forall a, b \in \mathbb{R}^d, \forall \lambda \in (0,1) : \qquad f(\lambda a + (1-\lambda)b) < \lambda f(a) + (1-\lambda) f(b)$$

An example for a strictly convex function is $f(x) = x^2$, as depicted in Figure 1b. Optimisation problems with a strictly convex objective functions are rather easy to solve, since there exists only one local minimum, which is also the global minimum.

Functions with different local and global optima

The third category we present contains optimisation problems whose functions have multiple local minima and at least one global minimum. As a consequence, an obtained local minimum is not necessarily a global optimum and therefore it might not be the solution to the problem. To give an example, the function depicted in Figure 1c has a global minimum at $x \approx -1.6$, but also a local minimum at $x \approx 1.6$. For this reason, solving optimisation problems of this category is usually difficult.

Solving methods

In general, it is desirable to obtain a solution for an unconstrained optimisation problem analytically, because one can rely on finding an exact global optimum when using analytical methods. For strictly convex functions, one can find the minimum by searching for the root of the derivative of the objective function, since only the single existing local minimum has a gradient of zero. For example, one can easily find the minimum of $f(x) = x^2$ by calculating the root of $f'(x) = 2x$, which is $x = 0$. For more complex functions however, obtaining a solution of the optimisation problem is analytically not feasible. For this reason, such problems have to be solved using algorithms that find numeric solutions. The main approach of numeric optimisation algorithms is described by Nocedal et al. in [8]. In general, numeric algorithms do not exploit analytical characteristics of the function, but instead only use properties of the objective function at a specific point, which includes e.g. the value of f and its gradient. For this reason, the main approach of optimisation algorithms is to start at an initial guess x_0 and to iteratively calculate a following iterate x_{k+1} with the use of information about f at the previous iterates $x_0, x_1, ..., x_k$. With this procedure, such an algorithm generates a sequence of iterates $\{x_k\}_{k=0}^{\infty}$. Optimisation algorithms are usually designed to converge to a local minimum, but have no guarantee for approximating one of the global optima. The sequence is terminated either once a sufficiently good approximation of the solution is found, or when the sequence converges, meaning that the distances between the last iterates are extremely small. If no solution exists, which is the case when no point minimises the objective function, the algorithm might not terminate, since for any iterate a point with a smaller objective value can be found. The core of the algorithms is the way they use information about f at the points $x_0, x_1, ..., x_k$ to compute x_{k+1}.

Nocedal et al. introduce two categories of optimisation algorithms: Line search and trust region strategies. The former first determines a line on which the next iterate is located, and then tries to find a point x_{k+1} such that $f(x_{k+1}) < f(x_k)$. Finding such an iterate is much simpler than the original minimisation problem, since a search only has to be performed along one dimension. A commonly used approach is the steepest descent method, where the direction is chosen to be the negative gradient, since it offers the locally steepest decent. Another important approach is to choose the Newton direction, which is calculated by finding a vector p minimising the second-order Taylor series approximation to $f(x_k + p)$. Line search methods also include quasi-Newton methods and conjugate gradient methods.

In contrast to line search methods, trust region strategies approximate the behaviour of the objective function f inside a trust region T around x_k using a simpler model function m_k. The model function is often computed using the first terms of the Taylor expansion, that is,

$$m_k(x_k + p) = f(x_k) + p^T \Delta f(x_k) + \frac{1}{2} p^T \Delta^2 f(x_k) p, \quad \text{where } x_k + p \in T.$$

Since the Hessian $\Delta^2 f(x_k)$ of the objective function can be hard to compute or might not be accessible, it can also be replaced by an approximation, which is for example calculated using the SR1 or BFGS formula [8]. The iterate x_{k+1} is then set to be the solution of the simpler problem $x_{k+1} = \underset{x \in T}{\operatorname{argmin}}\ m_k(x)$. Examples for trust region methods are the Dogleg method and Seihaug's approach [8].

2.2 Constrained optimisation problems

A constrained optimisation problem is an optimisation problem whose solutions have to satisfy a set of given constraints. Formally, they can be described as the following:

$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\operatorname{argmin}} && f(x) \\ &\text{subject to} && c_i(x) \geq 0,\ i \in \mathcal{I}. \end{aligned} \tag{2}$$

As before, $f\colon \mathbb{R}^n \to \mathbb{R}$ is our objective function and $n \in \mathbb{N}$ is the input dimension, whereas $c_i\colon \mathbb{R}^n \to \mathbb{R}, i \in \mathcal{I}$ are the inequality constraints with $\mathcal{I}$ denoting the set of all indices of the inequality constraints. Both f and all $c_i, i \in \mathcal{I}$ are assumed to be smooth. Further, we also assume that a solution for the given problem exists. We call any point satisfying all constraints *feasible*. Thus, the following defines the *feasible set*:

$$\mathcal{F} := \{x \in \mathbb{R}^n \mid c_i(x) \geq 0, i \in \mathcal{I}\}$$

In contradiction to unconstrained optimisation, where every local minimum x^* satisfies $\Delta f(x^*) = 0$ and $\Delta^2 f(x^*)$ being positive semidefinite, this is not necessarily true for solutions of constrained optimisation problems. The case of a nonzero gradient $\Delta f(\overline{x}) \neq 0$ of a solution $\overline{x}$ of the constrained problem can occur if all global minima x^* of f are not feasible. If this is the case, $\overline{x}$ might be located on the boundary of the feasible set, because all points with smaller values of f are infeasible. As a consequence, the negative gradient $-\Delta f(\overline{x})$ is directed towards the outside of F. In order to manipulate the objective function f in a way to allow the gradients of the solutions on the boundary of F to be zero, we introduce the Lagrangian Relaxation as described in [8]. Its core idea is to eliminate the gradients $-\Delta f(\overline{x})$ pointing towards the outside of the feasible set by adding the constraint values of the solution to the objective function, each of which is weighted with an individual Lagrangian multiplier. Since the negative gradients of the constraints point towards the feasible set F, they can eliminate the gradient of the objective function in the respective direction when multiplied by the correct nonnegative value. Formally, the Lagrangian for the constrained optimisation problem (2) is defined as follows:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{I}} \lambda_i c_i(x),$$

where $\lambda \geq 0$ is the vector of Lagrangian multipliers. First of all, minimising the Lagrangian Relaxation is much easier than solving the initial problem, since we can apply algorithms for unconstrained optimisation to the former problem. Furthermore, as mentioned before, this approach not only adds a bias towards feasible solutions, but more importantly allows the gradient on the boundary of the feasible region to equal zero in the direction of the gradient of the constraint. However, the magnitude of the weighted constraint gradient has to match the gradient of the objective function at the solution. This implies that setting the Lagrangian multipliers correctly is critical. Further, we have to note that $\mathcal{L}(x, \lambda) \leq f(x)$ for $x \in \mathcal{F}$, thus the Lagrangian provides a lower bound for f for all feasible points. As a consequence, we can obtain the sharpest possible lower bound by solving the Lagrangian Dual, that is,

$$\max_{\lambda} \min_{x} \mathcal{L}(x, \lambda).$$

However, according to Boyd et al. as claimed in [2], there usually is a duality gap between the solution of minimising the constrained problem and its Lagrangian Dual problem. For convex problems, the authors provide necessary and sufficient conditions for a duality gap of zero given that f and $c_i, i \in \mathcal{I}$ are convex.

Even though we have no guarantee to find the exact solution for problem (2) by solving the Lagrangian Dual, we introduce some optimality conditions which are more thoroughly explained in [8]. First, we define a set of indices of the active constraints, that is,

$$\mathcal{A}(x) = \{i \in \mathcal{I} \,|\, c_i(x) = 0\}.$$

We further define the *linear independence constraint qualification* (LICQ). It holds if the gradients of the active constraints $\{\Delta c_i(x^*), i \in \mathcal{A}(x^*)\}$ are linearly independent. This ensures that no gradient of an active constraint is zero.

Theorem 2.1 (*Karush-Kuhn-Tucker conditions - KKT*).
Let x^* be a solution of the constrained optimisation problem (2) *for which LICQ holds. Then there exists a Lagrange multiplier vector λ^*, such that*

$$\begin{aligned} \Delta_x \mathcal{L}(x^*, \lambda^*) &= 0, && && \text{(3a)} \\ c_i(x^*) &\geq 0, && \forall i \in \mathcal{I}, && \text{(3b)} \\ \lambda_i^* &\geq 0, && \forall i \in \mathcal{I}, && \text{(3c)} \\ \lambda_i^* c_i(x^*) &= 0, && \forall i \in \mathcal{I}. && \text{(3d)} \end{aligned}$$

The proof to Theorem 2.1 is quite complex and can be found in [8]. However, this statement is powerful, since it implies that for the right Lagrangian multipliers λ, solutions of the constrained optimisation problem have a gradient of zero in the relaxed problem and can therefore be found by unconstrained optimisation solving methods. Moreover, it provides an optimality test for solutions that minimise the Lagrangian, since the KKT conditions (3) are necessary conditions for solutions of problem (2).

2.3 Solving strategies

In the following, we present two strategies to solve constrained optimisation problems. We first discuss the Penalty Method and continue with the Augmented Lagrangian Method. While the first method penalises constraint violations, the latter is based on solving the previously introduced idea of Lagrangian Relaxation.

Penalty Method

The first approach we introduce is the Penalty Method. As the name suggests, we penalise constraint violations, thus favouring feasible solutions. In particular, the values of each violated constraint is added quadratically to the objective function. This leads to a bias towards the feasible region. In contrast to the Lagrangian Relaxation, where the Lagrangian multipliers weight the constraints linearly to eliminate the gradients at the boundary, the Penalty Method does not take constraints into account that are satisfied at all. On the one hand, this leads to the advantage of leaving values of f in the feasible region unchanged, on the other hand, gradients of solutions on the boundary of the feasible region are not manipulated either, which means that even solutions of the constraint optimisation problem on the boundary can have a gradient other than zero and are therefore no local minima. This problem is tackled by solving a series of minimisation problems while increasing the weight of the constraint penalisation, such that the global optimum of the function including the penalty converges to the global optimum of the objective function. Formally, each of the minimisation problems is of the following form:

$$\operatorname*{argmin}_{x \in \mathbb{R}^n} Q(x, \lambda_k), \qquad Q(x, \lambda) := f(x) + \lambda \sum_{i \in \mathcal{I}} (c_i^-(x))^2$$

where $\{\lambda_k\}_{k=1}^{\infty}$ is a monotonically increasing series of weights with $\lim_{k\to\infty}\lambda_k = \infty$ and $c_i^-(x) = \min(0, c_i(x))$. Intuitively, this means that only violated constraints are penalised and as the penalty weight λ increases, feasible solutions are favoured even more with each iteration.
In the following, we prove that the Penalty Method converges to an optimal solution.

Theorem 2.2. *Let* $x_k = \operatorname{argmin}_{x\in\mathbb{R}^n} Q(x, \lambda_k)$*. Then every limit point* $x^* = \lim_{k\to\infty} x_k$ *is a solution of the constrained minimisation problem* (2)*.*

Proof. The proof is based on the proof given in [8] with some adaptations to fit our problem formulation.
Let $\overline{x}$ be a solution of the constrained optimisation problem as defined in equation (2). Thus, the following holds:

$$f(\overline{x}) \leq f(x), \qquad \forall x \in \mathcal{F}$$

Since x_k minimises $Q(x, \lambda_k)$, we have

$$f(x_k) + \lambda_k \sum_{i\in\mathcal{I}} (c_i^-(x_k))^2 \leq f(\overline{x}) + \lambda_k \sum_{i\in\mathcal{I}} (c_i^-(\overline{x}))^2 = f(\overline{x}) \tag{4}$$

From this expression, we can now obtain an upper bound on the constraint values, that is,

$$\sum_{i\in\mathcal{I}} (c_i^-(x_k))^2 \leq \frac{1}{\lambda_k}(f(\overline{x}) - f(x_k)). \tag{5}$$

Let x^* be a limit point of $\{x_k\}_{k=1}^{\infty}$, thus there exists an infinite subsequence $\mathcal{K}$ with $\lim\limits_{k\in\mathcal{K}} x_k = x^*$. Then we can apply the limit on both sides of inequality (5) to obtain the following:

$$\sum_{i\in\mathcal{I}} (c_i^-(x^*))^2 = \lim_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} (c_i^-(x_k))^2 \leq \lim_{k\in\mathcal{K}} \frac{1}{\lambda_k}(f(\overline{x}) - f(x_k)) = 0,$$

since $\lim\limits_{k\in\mathcal{K}} \lambda_k = \infty$. This implies that all constraints are satisfied and the obtained solution x^* is located in the feasible set F. Thus, in order for x^* to be a solution of the constrained optimisation problem, we need to show that it minimises the objective function in the feasible set. This can be done by showing $f(x^*) \leq f(\overline{x})$, since $\overline{x}$ is a solution for the constrained minimisation problem and thus any point with the same or a lower value of f is also a solution. To do so, we take the limit $k \to \infty, k \in \mathcal{K}$ in inequality (4):

$$f(x^*) \leq f(x^*) + \lim_{k\in\mathcal{K}} \lambda_k \sum_{i\in\mathcal{I}} (c_i^-(x_k))^2 \leq f(\overline{x})$$

$\square$

In general, the Penalty Method allows us to solve the constrained optimisation problem using methods for unconstrained optimisation on a series of minimisation problems. However, despite the existence of an analytical proof, a numeric conversion to the optimum of the constrained optimisation problem is not guaranteed. As Nocedal et al. point out in [8], the reason for this is that gradient based solving methods suffer from ill-conditioning when applied to $Q(x, \lambda)$ with large values for λ, which cannot be avoided when trying to converge to a minimising and feasible solution.

Augmented Lagrangian Method

In the following, we introduce the Augmented Lagrangian Method (ALM). It tackles the problem of ill-conditioning as it occurs in the Penalty Method by adding the linear constraint term of the Lagrangian Relaxation with explicit estimates of the Lagrangian multipliers. We explain the case of having only a set of equality constraints $c_i(x) = 0, i \in \mathcal{E}$, where $\mathcal{E}$ is the set of indices of the equality constraints. This means the problem to solve is given by

$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\operatorname{argmin}} && f(x) \\ &\text{subject to} && c_i(x) = 0,\ i \in \mathcal{E}. \end{aligned} \tag{6}$$

The general case including inequalities can be solved by introducing slack variables; the interested reader is referred to [8]. The Augmented Lagrangian Method tries to solve the constrained problem (6) by solving a series of minimsation problems of the form

$$x_k = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \mathcal{L}_A(x, \lambda_k, \mu_k), \qquad \mathcal{L}_A(x, \lambda, \mu) := f(x) - \frac{1}{2} \sum_{i \in \mathcal{E}} \lambda_i \, c_i(x) + \mu \sum_{i \in \mathcal{E}} c_i^2(x),$$

where $\{\mu_k\}$ is an increasing series, and the Lagrangian multipliers are updated according to the formula

$$\lambda_i^{k+1} = \lambda_i^k - \mu_k \, c_i(x_k). \tag{7}$$

Assuming that the Lagrangian multipliers converge, we can see that the constraint values are now much smaller than $\frac{1}{\mu_k}$, since

$$c_i(x_k) = -\frac{1}{\mu_k}(\lambda_i^{k+1} - \lambda_i^k), \qquad i \in \mathcal{E}.$$

This means that in order to decrease the constraint values to close to zero, we do not purely rely on increasing μ, but also achieve smaller constraint violations due to the convergence of the Lagrangian multiplier estimates. This makes it possible to show that ALM converges without increasing μ to an extremely large value as it is required for the Penalty Method [8].

Physical Projection

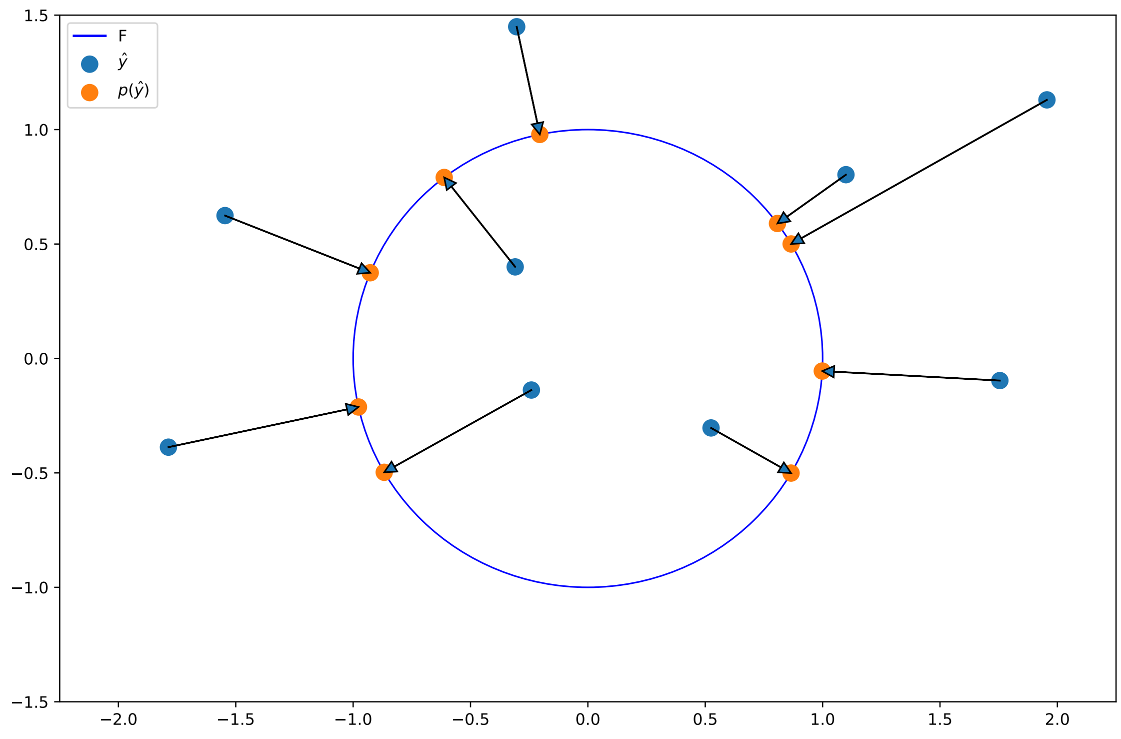


Figure 2: Visualisation of an euclidean orthogonal projection of points to the unit circle.

In addition to the Penalty Method and the Augmented Lagrangian Method, we want to introduce an approach we call Physical Projection, which is similar but not to be confused with the method of Projected Gradient Descent as described in [3]. The Physical Projection Method can be applied when using a deep learning model parameterised by θ to approximate a function $f : X \to Y$ representing the ground truth with a set of physically feasible points $F \subseteq Y$. Usually, such a model is trained to represent a function $g_\theta : X \to Y$ that approximates f. However, since we know that any output of the ground truth f is feasible, we project any prediction $g_\theta(x), x \in X$ to a point in the feasible set F using an orthogonal projection function $p : Y \to F$ of the form

$$p(y) = \underset{\hat{y} \in F}{\operatorname{argmin}} ||\hat{y} - y||, \qquad y \in Y,$$

where $|| \cdot ||$ denotes any norm. This means that any prediction that is already feasible is not changed and any infeasible prediction is projected to the closest feasible point with respect to the given norm. An example for a projection that maps points of $\mathbb{R}^2$ to the closest respective points on the unit circle with respect to the euclidean norm is depicted in Figure 2. Using such a projection, the outputs of the trained model are then calculated by $p(g_\theta(x)), x \in X$.

However, for many real world applications, the projection is either not known or analytically not feasible. For such problems, one can utilise the approach of training an additional model $\hat{p}_\theta : Y \rightarrow Y$ to learn the true projection p. For any point, it should output an approximation of the closest feasible point. Since the projection itself is again a constrained optimisation problem, we can find an approximate solution by transforming it to an unconstrained problem. In particular, a simple corresponding unconstrained optimisation problem is obtained by adding a weighted physical loss $\mathcal{L}_{PHY}$ measuring the feasibility of a solution x to the distance:

$$\tilde{p}_\theta(y) = \underset{\hat{y}}{\operatorname{argmin}}(||\hat{y} - y|| + \eta \mathcal{L}_{PHY}(\hat{y})), \tag{8}$$

where η is the weight of the physical loss. Since the physical loss is measuring the violation of the physical constraints, it can be calculated as it is done when using the Penalty Method, that is, $\mathcal{L}_{PHY}(\hat{y}) = \sum_{i \in \mathcal{I}} (c_i^-(\hat{y}))^2$. By learning the solution of the unconstrained problem (8), the model optimises both the distance between the input and the output, and the violation of physical constraints.

The additional approach of learning the projection with an individual model is not applied in this thesis and therefore remains to be explored by future studies.

3 Experiment

In the following, we explain the framework we use to evaluate the approaches introduced above. After describing the experiment on which our models are trained, we illustrate different network architectures and hyperparameters. This is followed by a specification of the methods we apply to incorporate physical constraints into the learning process.

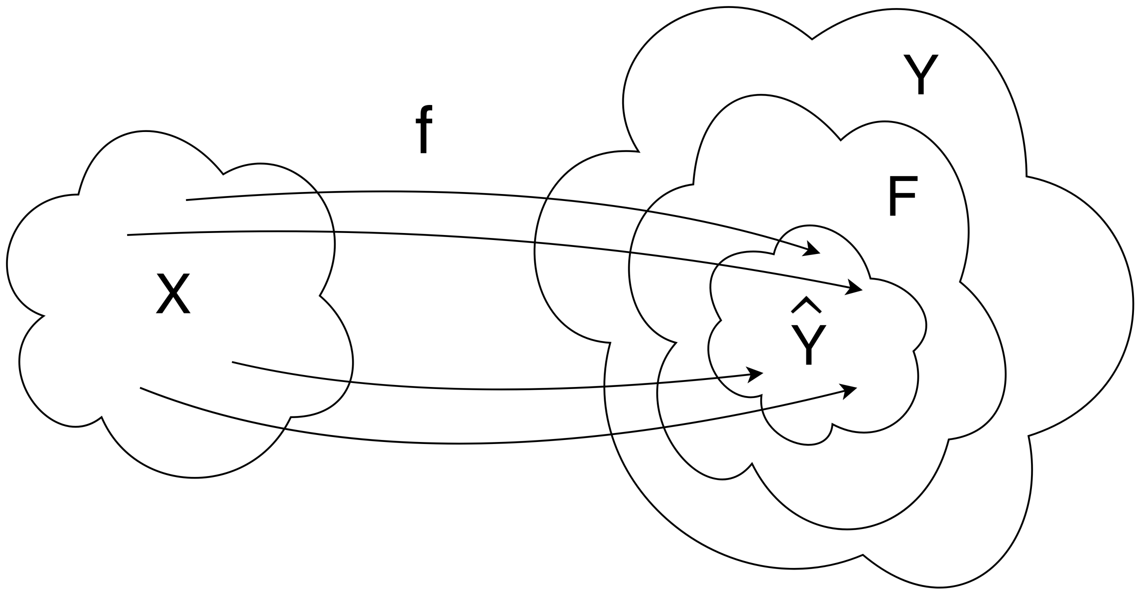


Figure 3: Visualisation of a function f mapping from X to $\hat{Y}$. F denotes the physically feasible set and Y a larger space for which we know that $\hat{Y} \subseteq Y$.

Before choosing an experiment to test our approaches on, we need to look at the structure of a learning problem. In general, a learning problem is the task of creating a function $\hat{f}_\theta : X \to Y$ with a model parameterised by θ that approximates the ground truth function $f : X \to \hat{Y}$, which maps given inputs of the domain X to the corresponding output in the codomain $\hat{Y}$. An example for a ground truth function is visualised in Figure 3. Usually, we do not have access to the ground truth f, but instead try to learn it with the help of a set of data points $D \subseteq \{(x, y), x \in X, y \in \hat{Y} | f(x) = y\}$. Since we know that any ground truth satisfies all physical constraints, the set of outputs $\hat{Y}$ of f is a subset of the set of all feasible points F, meaning that $\hat{Y} \subseteq F$. However, when trying to learn the function $\hat{f}_\theta : X \to \hat{Y}$, we usually do not know the set of outputs $\hat{Y}$. Therefore, we start with the representation $\hat{f}_\theta : X \to Y$, where Y is a much larger space for which we know that $\hat{Y} \subseteq F \subseteq Y$. To give an example, we look at the following function:

$$\begin{aligned} g :& [0, \pi] \to [0, 1] \\ x \mapsto& \sin(x) \end{aligned} \tag{9}$$

Even though the function g only maps values of the domain space $X = [0, \pi]$ to $\hat{Y} = [0, 1]$, feasible solutions contain all values between -1 and 1, namely $F = [-1, 1]$, since we know that the function $\sin$ only outputs values in this range. Note that for this example, one could start with a model function $\hat{f}_\theta : X \to \hat{Y}$ parameterised by θ mapping only to $\hat{Y} = [0, 1]$, for example by applying the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ to any predicted output. However, the set of ground truth outputs of real world problems is usually very complicated, which makes it impossible to initialise the model function $\hat{f}_\theta$ to only output values in $\hat{Y}$. For this reason, when trying to train a model $\hat{f}_\theta$ without any prior knowledge about the ground truth function f, we start by constructing a function $\hat{f}_\theta : X \to Y$ that maps inputs to the larger space Y. In our example of the function g, one would initially construct a model function $\hat{f}_\theta : [0, \pi] \to \mathbb{R}$ mapping inputs to the real numbers.

By incorporating physical constraints into the training process of deep learning models, we try to reduce the distance between the predictions $y_{pred} \in Y$ and the correct values $\hat{y}_{true} \in \hat{Y}$ by training the model to output values in or close to the feasible region F.

Since it can be difficult to break down the effects of incorporating knowledge about physical constraints on a deep learning model, the previously introduced methods have to be applied to a simple, yet not trivial problem. For measuring how well predictions align with the given physical constraints, we introduce the term *realistic* to describe predictions with no or small constraint violations. By analysing the behaviour of our model, we aim to show that incorporating physical constraints contributes in the following ways:

- Improve model performance,
- Create more realistic predictions,
- Decrease required amount of learning data.

A well fitting problem that satisfies both simplicity and the existence of physical constraints is provided by the task of learning a rotation. A rotation is a function that takes coordinates of a point together with rotation angles as input and outputs the coordinates of the rotated point. The function can be described as a multiplication of a rotation matrix, which only depends on the rotation angle, and the point to be rotated. Rotations are invariant with respect to the L2-norm, meaning that the norm of the rotated point equals the norm of the input point. In addition, the determinant of the rotation matrix equals one. These two properties serve as physical constraints. Since the rotation problem satisfies the required characteristics, we use it to evaluate the previously introduced methods for incorporating physical constraints. The rotation problem itself is explained in more detail in the following section.

3.1 Problem formulation

In order to show the desired improvements, we apply the introduced methods to the rotation problem in both two and three dimensions. Formally, a rotation function maps a point $\vec{x}$ and rotation angles $\vec{\alpha}$ to a target point $\vec{y}$:

$$\begin{aligned} rot : \mathbb{R}^d \times [-\pi, \pi]^{d-1} &\to \mathbb{R}^d \\ (\vec{x}, \vec{\alpha}) &\mapsto rot(\vec{x}, \vec{\alpha}) = \vec{y}, \end{aligned} \tag{10}$$

where d is the dimension of the domain space. For example, the rotation in two dimension can be described in the following way:

$$\begin{aligned} rot_{2D} : \mathbb{R}^2 \times [-\pi, \pi] &\to \mathbb{R}^2 \\ (\vec{x}, \vec{\alpha}) &\mapsto rot_{2D}(\vec{x}, \vec{\alpha}) = R_{2D}(\vec{\alpha})\, \vec{x}, \end{aligned} \tag{11}$$

$$\text{where } R_{2D}(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}.$$

For three dimensions, we will consecutively apply a rotation around the z-axis and the y-axis, both counterclockwise when looking towards the origin. This means that the rotation function maps the point x and the rotation angles $\vec{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}$ according to the following:

$$\begin{aligned} rot_{3D} : \mathbb{R}^3 \times [-\pi, \pi]^2 &\to \mathbb{R}^3 \\ (\vec{x}, \vec{\alpha}) &\mapsto rot_{3D}(\vec{x}, \vec{\alpha}) = R_{3D}(\vec{\alpha})\, \vec{x} = R_y(\alpha_2) R_z(\alpha_1)\, \vec{x} \end{aligned} \tag{12}$$

$$\text{where } R_z(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } R_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix}.$$

Regarding physical constraints, we know that the determinant of any rotation matrix equals one and thereby preserves the norm of any point. Therefore, we have the following two physical constraints for our experiment:

$$\det(R(\vec{\alpha})) = 1, \qquad \forall \vec{\alpha} \in [-\pi, \pi]^d, \forall R \in \{R_{2D}, R_{3D}\} \tag{13a}$$

$$||R(\vec{\alpha})\vec{x}||_2 = ||\vec{x}||_2, \qquad \forall \vec{\alpha} \in [-\pi, \pi]^d, \forall \vec{x} \in \mathbb{R}^d, \forall R \in \{R_{2D}, R_{3D}\}. \tag{13b}$$

The proofs for the determinant and norm constraint for two (resp., three) dimensions can be found in the appendix in (18) (resp., (19)) and (20) (resp., (21)), respectively.

3.2 Training the Deep Learning Model

Using a deep learning model, we can now learn the rotation function rot as defined in equation (10) with the use of training data. Since the available training data is the biggest limitation for real life applications, we focus on improving the performance of neural networks using training datasets of limited sizes with prior knowledge about physical constraints. Therefore, we are mainly interested in the performance of our deep learning models with respect to the number of training datapoints. By comparing the model trained solely on the training data with the ones utilising the previously introduced methods, we try to show that incorporating the knowledge about physical principles increases both the performance and how well the predictions align with the physical constraints for limited amounts of training data.

In the following, we explain technical details about the training dataset creation and the training process. For the remainder of this thesis, N_{train} denotes the number of points included in our training dataset. The training data itself is denoted X_{train}. It contains N_{train} elements of the set $\mathbb{R}^d \times [-\pi, \pi]^{d-1}$. The corresponding correct target points are denoted y_{train}.

The points to be rotated for the $2D$- and $3D$-Rotation are drawn uniformly from the unit circle and the unit sphere, respectively. The rotation angles are chosen uniformly from the set $[-\pi, \pi]^{d-1}$. Note that for three dimensions, this leads to a bias towards the poles, meaning that the target point is more likely to be either close to the original point or to the point located exactly on the opposite side of the sphere.

When training a neural network to learn the rotation function, we solve the following minimisation problem:

$$\underset{\theta}{\operatorname{argmin}} \; \mathcal{L}(f_\theta(X_{train}), y_{train}),$$

where f_θ is the function represented by the model parameterised by θ and $\mathcal{L}$ is a loss function measuring the dissimilarity between the model predictions $f_\theta(X_{train})$ and the true target points y_{train}. From this point on, the model predictions $f_\theta(X_{train})$ for a fixed parameter set θ will be denoted by $\hat{y}$. A commonly used loss function is the Mean Squared Error (MSE), which is calculated in the following way:

$$\mathcal{L}_{MSE}(\hat{y}, y) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{d} \sum_{i=1}^{d} (\hat{y}_{n\,i} - y_{n\,i})^2, \qquad \text{where } y, \hat{y} \in \mathbb{R}^{N \times d} \tag{14}$$

All models will be trained using the MSE between the predictions and the target points. Minimising the MSE loss function fits well for our experiment, since it is closely related to the euclidean distance, except that distances are squared. This leads to high values for distant predictions and thus penalises higher variance of the errors of the predictions, which is desirable.

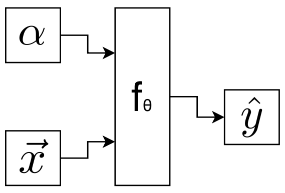
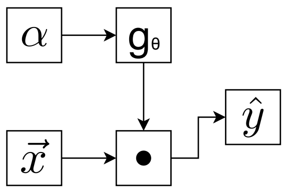
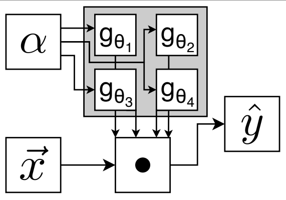
3.3 Performance evaluation

For the same reason as we chose the MSE Loss as the training loss, we also evaluate the performance of the introduced models and methods according to the MSE. Despite the MSE being a good measure for the distance between the predictions and the target point, it does not measure how well the predictions of the model align with the given physical constraints. In order to check how realistic the predictions are, we explicitly compare their norms and determinants.

3.4 Network architectures

In the following, we introduce three different model architectures parameterised with weights θ to learn the rotation function. The most important characteristics are depicted in Table 1.

Table 1: Model architectures for two dimensions. α is the rotation angle, $\vec{x}$ denotes the input point and $\hat{y}$ is the predicted point. The operator $\boxdot$ represents matrix-vector multiplication.

	Model 1	Model 2	Model 3
Structure			 α g_{θ_1} g_{θ_2} g_{θ_3} g_{θ_4} $\vec{x}$ $\hat{y}$
Network function	$f_\theta : \mathbb{R}^3 \to \mathbb{R}^2$	$g_\theta : \mathbb{R} \to \mathbb{R}^{2\times 2}$	$g_{\theta_i} : \mathbb{R} \to \mathbb{R}, i \in \{0,1,2,3\}$
Layer sizes	$3 \to 16 \to 16 \to 16 \to 2$	$1 \to 100 \to 4$	$[1 \to 50 \to 1] \times 4$
Activation	Tanh	Sigmoid	Sigmoid
Parameters	642	604	604

Model 1: The first model is a neural network directly approximating f_θ. Thus, it takes a point of dimension d and angles of dimension $d-1$ as input and predicts a target point of dimension d. The network has three hidden layers with 16 nodes each. In each of the hidden layers, the activation function $\tanh$ is applied.

For **Model 2** and **Model 3**, we make use of the knowledge that a rotation is the result of multiplying a rotation matrix, which only depends on the angles α, with the given point $\vec{x}$. Consequently, the networks of the next models represent the function $g_\theta : [-\pi, \pi]^{d-1} \to \mathbb{R}^{d\times d}$, which in turn is used to create the final predictions in the following way:

$$\hat{y} = g_\theta(\alpha)\,\vec{x}, \tag{15}$$

where $\alpha \in [-\pi, \pi]^{d-1}$ is the rotation angle and $\vec{x} \in \mathbb{R}^d$ the point to be rotated.

Model 2: Based on this approach, the second model predicts the rotation matrix. In particular, it maps $d-1$ input angles to a matrix of size $d \times d$. This is done using one hidden layer of size 100 with the Sigmoid activation function.

Model 3: Similar to the previous model, we also try to solve the problem using an independent neural network for each of the $d \times d$ matrix entries. Each of these networks maps the rotation angles of size $d-1$ to a single real number, which is interpreted as a single matrix entry. For each of these networks, we use one hidden layer of size 50 with the Sigmoid activation function.

3.5 Hyperparameters

All layers of the neural networks include bias and we do not apply dropout. In order to achieve undistorted comparisons of the models, they were designed to have between 600 and 650 parameters each for the two-dimensional problem. For all experiments, we decide to use the Adaptive Moment Estimator "Adam" as our optimiser, since it empirically appeared to perform well in practice and is favourable to other known adaptive learning-method algorithms [9]. Furthermore, we use a learning rate of 5×10^{-5} and train each model using $50\,000$ iterations. Both of these parameters are empirically estimated, a comparison of using different learning rates on Model 3 is displayed in Figure 12 in the appendix. For all experiments, we use a batch size of 512 and shuffle the training set at the start of each epoch.

3.6 Metrics

In the following, we introduce different loss functions to measure the desired properties of the predictions. The choice of these loss functions is critical, since they determine the objective function which is minimised in the training process. In particular, the loss functions need to measure the predictions' accuracy and physical feasibility.

Accuracy

As explained previously, the MSE Loss defined in equation (14) is a well fitting function to measure the accuracy of the predictions, since it is the sum of the square distances between the predictions and the corresponding target points. Hence, training on the MSE Loss aims to reduce this distance.

Phyiscal feasibility

In order to train our models to learn the physical constraints, we introduce loss functions specifically designed to measure how realistic the model predictions are. In general, our physical loss functions map the predicted points and, if applicable, the predicted rotation matrix to a real number, which is interpreted as the violation of the physical constraints. Thus, the physical loss functions are of the form

$$L_{PHY} : \mathbb{R}^d \times \mathbb{R}^{d \times d} \to \mathbb{R}.$$

Determinant constraint In order to measure how well the predicted rotation matrices align with the constraint of the determinant being equal to one (13a), we use the following function as our physical loss:

$$L_{DET}(\hat{y}, \hat{R}) = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} (\det(\hat{R}_n) - 1)^2,$$

where $\hat{R}$ are the predicted rotation matrices computed by applying g_θ to each element of X_{train}. This can also be described as the MSE Loss between the vector of the predicted matrices' determinants and a vector of the same size filled with ones.

Norm constraint In order to incorporate the constraint that the norm of any prediction needs to be one (13b), the following physical loss function is applied:

$$L_{NORM}(\hat{y}, \hat{R}) = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} (||\hat{y}_n||_2 - 1)^2,$$

where $\hat{y}$ denotes the points predicted by the model. Similar to the Determinant loss, the Norm loss can be interpreted as the MSE Loss between the vector of the norms of the predictions and a vector of ones of the same size.

3.7 Solving methods

In this section, we will introduce the methods we apply to incorporate the physical loss functions into the models. As a baseline for comparisons, we train each model using solely the MSE Loss on the training data.

Penalty Method

The first and simplest version of the Penalty Method we apply is setting a fixed weight $\lambda > 0$ and solving a single minimisation problem, that is,

$$\underset{\theta}{\operatorname{argmin}} \, (\mathcal{L}_{MSE}(\hat{y}_\theta, y_{train}) + \lambda \cdot L_{PHY}(\hat{y}_\theta, g_\theta(X_{train}))).$$

In addition, we also solve a series of minimisation problems of the type above with exponentially increasing λ, but using the solution of the last iteration as a warm start for the next one. In particular, we introduce a multiplier $\mu > 1$ such that the weight of the physical loss in the i-th iteration is given by $\lambda_i = \mu^i \, \lambda_0$. Each minimisation is stopped as soon as the norm of the gradient is below a certain threshold.

Augmented Lagrangian Method

As we do for the Penalty Method, we also apply two different versions of the Augmented Lagrangian Method. Both versions minimise a fixed number of problems and update the weights of the linear constraint terms according to rule (7). However, the first one only computes a fixed number of epochs with a constant weight λ for the physical loss for each minimisation problem.

The second version was suggested by Bertsekas in [11]. In addition to increasing the weight of the squared constraint term, he proposes to stop the minimisation of the k-th problem as soon as the norm of the gradient is smaller than a threshold τ_k computed according to the following equation:

$$\tau_k = \min(\epsilon_k, \gamma_k ||c(x_k)||_2),$$

where $\{\epsilon_k\}$ and $\{\gamma_k\}$ are two sequences decreasing to 0 and $c(x_k)$ is the value of the constraint violation at the current solution. In our experiment, $c(x_k)$ is the difference of the norms / determinants of the predicted points / rotation matrices and one. Intuitively, this approach spends more ressources on minimising a problem if the predictions of the current solution already align well with the physical constraints and otherwise focuses on learning the constraints first.

Physical Projection

As a third approach, we apply the Physical Projection. In particular, each prediction in the training process is projected to the closest feasible prediction. When applying it to the norm loss, we divide the predicted point $\hat{y}$ by its norm and thus get a prediction $\hat{y}'$ located on the unit circle or unit sphere. Formally, predictions are calculated the following way:

$$\hat{y}' = \frac{\hat{y}}{||\hat{y}||_2}. \tag{16}$$

For the determinant, we only divide the matrix by the d-th root of the determinant if the latter is positive, otherwise we do not change the matrix. Therefore any predicted rotation matrix with a positive determinant is projected to a new matrix with a determinant of one. Formally, we update each predicted matrix $\hat{R}_n$ according to the following rule:

$$\hat{R}_n' = \begin{cases} \frac{\hat{R}_n}{\sqrt[d]{\det(\hat{R}_n)}}, & \text{if} \det(\hat{R}_n) > 0 \\ \hat{R}_n, & \text{else} \end{cases}, \qquad n = 1, ..., N_{train}. \tag{17}$$

Since matrices with negative determinants are far away from the true rotation matrix, they should not occur often or if so, be corrected by solely training on the MSE-Loss for the prediction and the target point.

3.8 Statistical measurements

Each experiment will be tested on 20 different seeds used to generate the training data, with a few exceptions due to time limitations. If not otherwise specified, we compare the performance of the models according to the mean of the MSEs of the individual runs. We prefer the mean over the median, because we are also interested in the performance of outliers that lead to poor results. The test set on which each model's performance is calculated consists of 4096 data points sampled in the same manner as the training dataset.

4 Results

In this section, we compare the performance of the previously introduced models and methods and illustrate the results. They provide insights on the effects of incorporating knowledge about physical constraints and some limitations. All experiments have been implemented in Python using PyTorch. The implementation to reproduce the results can be found at https://github.com/KorbinianBstrtr/BAThesis_abstreik.git.

4.1 Model architecture

As described in the section 3.4, we first compare the three introduced network architectures. The performance of the models is displayed in Figure 4.

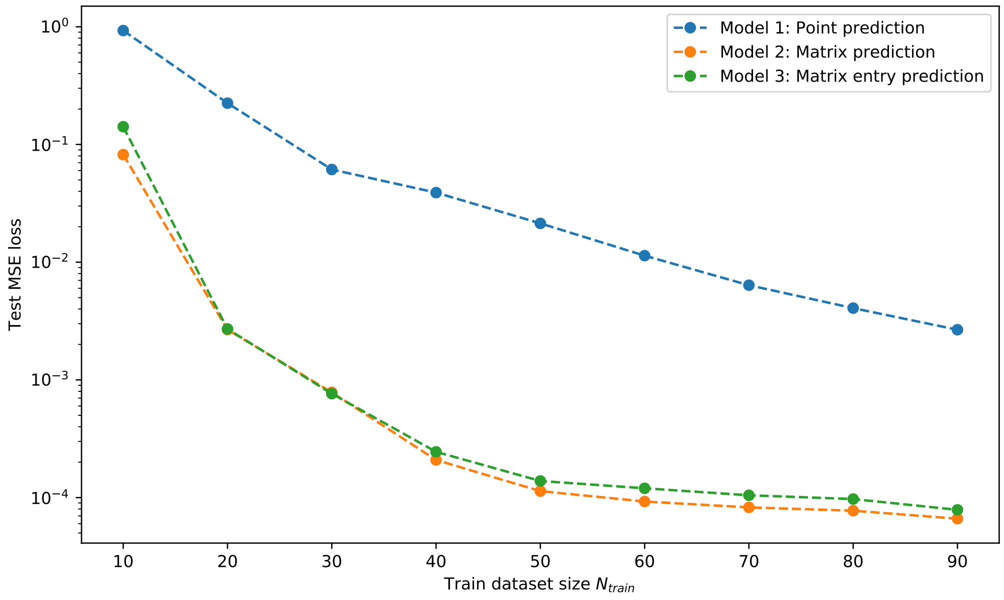


Figure 4: Different model architectures

We can see that while the difference between the behaviours of Model 2 and 3 is quite small, both of them perform significantly better than Model 1. Even when trained on 90 data points, Model 1 only achieves an error of 8×10^{-3}, whereas Model 2 and 3 achieve a lower value of 2.7×10^{-3} with only 20 data points. This aligns with our expectations, since the latter two models already make use of the knowledge that a rotation can be represented as the multiplication of the point with a matrix that only depends on the rotation angle. Apparently, this theoretical knowledge is very helpful, which is why Model 2 and Model 3 achieve significantly smaller error than Model 1. Even though the two matrix models Model 2 and 3 perform well and almost the same for 20 and 30 data points, Model 2 finds a slightly better solution for most other amounts of training data. For example, with 10 data points, Model 2 achieves an error of 8.2×10^{-2}, which is 41% less than the performance of Model 3, which only reaches 1.4×10^{-1}. The observed behaviour can be explained by looking at the structural difference of Model 2 and 3. Model 2 only uses one network with a single hidden layer that is fully connected with all output nodes which represent the rotation matrix entries. In contrast, Model 3 has an independent neural network for each matrix entry. As a consequence, the backpropagation in Model 3 for every network is done independently of the other neural networks. However, the matrix entries of the rotation matrix correlate heavily with each other, since the $\cos$ entries are exactly the same and the $\sin$ entries only differ in the sign. Therefore, trying to learn these entry values independently, as Model 3 does, constitutes a major disadvantage which leads to worse performance.

Since Model 1 performs drastically worse than Model 2 and 3 for the reasons mentioned above, we focus our experiments on the latter two models. In addition, due to time limitations, most of the following experiments are only carried out using Model 3. Future studies could check if the same behaviour occurs when applying the introduced methods to Model 2. Furthermore, since Model 3 already reaches a Test MSE Loss of less than 3×10^{-3} with 20 training data points, but still performs poorly with only 10 points, the majority of the following analysis focuses on the interesting region of 10 to 20 as the size of the training dataset.

However, we need to be careful not to attribute performance improvements to the incorporation of physical constraints if the same performance can also be achieved differently. Since both the Norm and the Determinant loss act as a form of regularisation, we need to check the effect of applying commonly used other types of regularisation. Hence, we apply L2-Regularisation with a wide range of different values for the weight decay. When conducting this experiment, we can observe no consistent improvement of the model's performance for any regularisation weight. The results can be looked up in Figure 13 in the appendix.

4.2 Penalty Method

Fixed physical loss weight

First, we apply the Penalty Method using a fixed weight λ for the determinant physical loss $\mathcal{L}_{DET}$ and therefore only solve a single minimisation problem during the training process. The results obtained by this method performing $50\,000$ epochs on the 2D-Rotation problem using Model 3 are shown in Figure 5. When looking at these results, we immediatly observe the method's robustness to different weights of the physical loss, since small weights in a range of at least two orders of magnitude, namely $\lambda \in [0.01, 1]$, lead to significantly improved performance. For smaller values of λ, the test error converges to the one of $\lambda = 0$, whereas values of $\lambda > 1$ lead to a heavy focus on satisfying the physical constraint and thus leads to poor results compared to smaller values and $\lambda = 0$.

Moreover, we can see that the choice $\lambda = 0.1$ yields the smallest error on the test dataset for most training dataset sizes. It is also important to note that the performance improvement compared to the baseline is significant. For example, with only 20 training data points, the physically trained model achieves a Test MSE Loss of 8×10^{-4}, whereas the model trained solely on the MSE Loss achieves only a value of 2.7×10^{-3}, thus reducing the error by more than 70%.

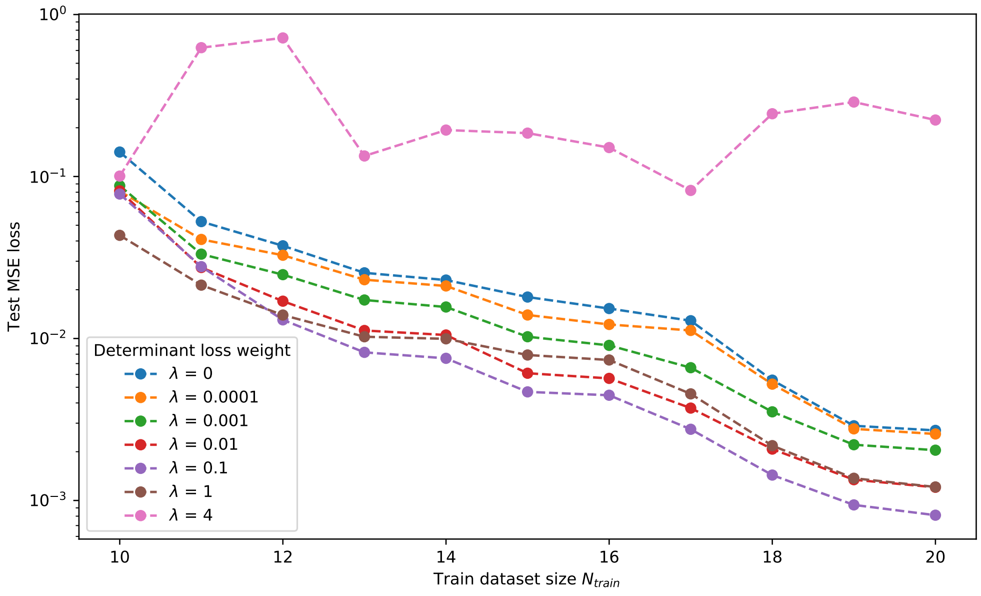


Figure 5: Different weights λ for $\mathcal{L}_{DET}$

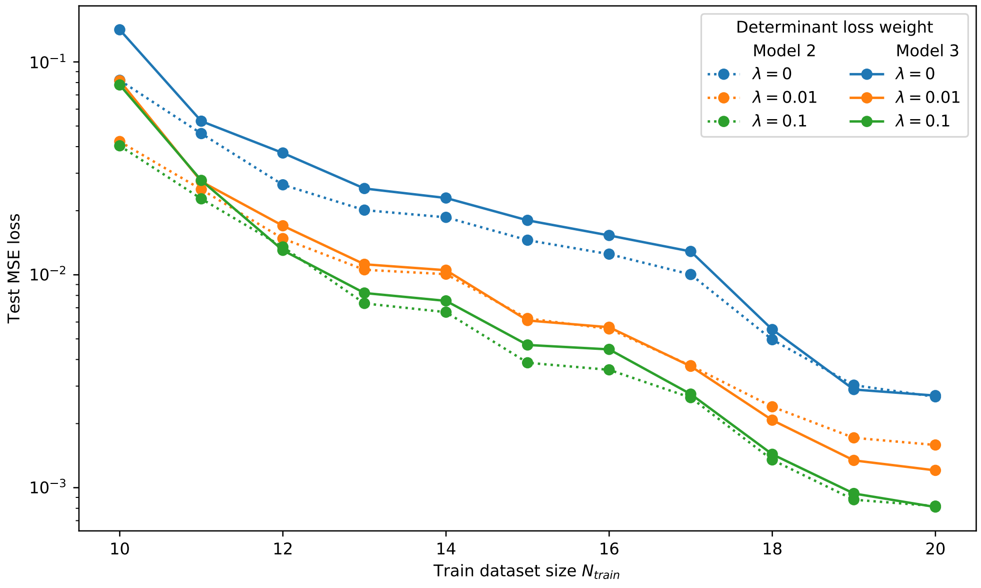


Figure 6: Performance of the Fixed Penalty Method applied to Model 2 and Model 3 using different weights for the determinant loss.

Interestingly, utilising the Penalty Method leads to higher performance even for larger train datasets. This can be seen in Figure 14 in the appendix, displaying the results of an experiment we conducted on a wider range including up to 100 training data points.
We observe similar qualitative behaviour when applying the method for the 3D-rotation, as displayed in Figure 15 in the appendix.
In order to check whether similar effects can be observed when using another model, we additionally apply the Fixed Penalty Method with the weights $\lambda = 0.01$ and $\lambda = 0.1$ to Model 2. The performance together with the one of Model 3 is depicted in Figure 6. When looking at these results, we can clearly see the same qualitative behaviour for the different physical loss weights compared to the baseline. For both models, applying the Penalty Method with the weights of $\lambda = 0.01$ and $\lambda = 0.1$ improves performance significantly compared to not incorporating the physical constraints.

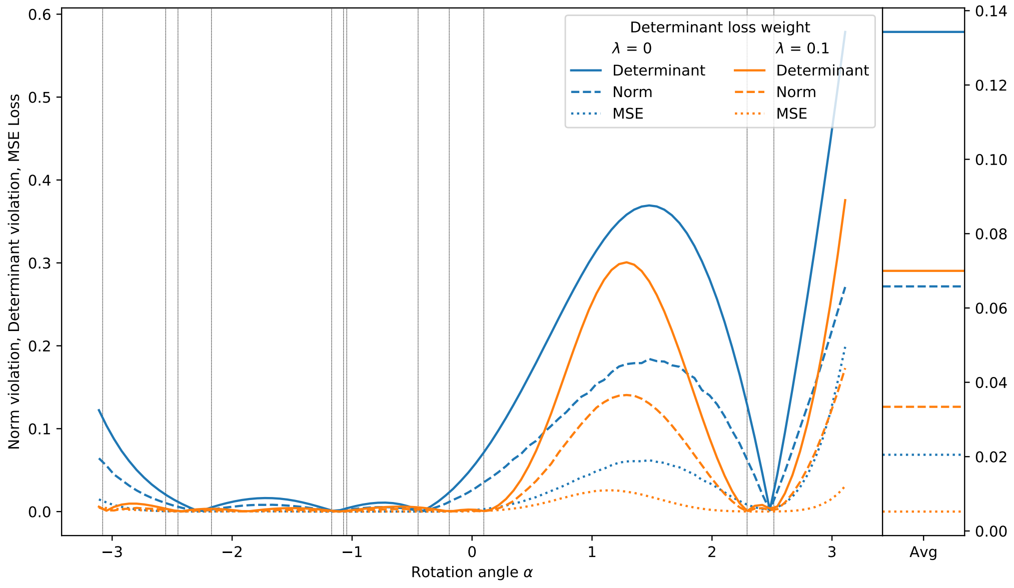


Figure 7: Determinant and norm violations of predictions for $N_{train} = 12$ using Model 3. Gray vertical lines represent the angles of the training points.

Since we are not only interested in the overall test performance, but also in the physical feasibility of the predictions, we further need to analyse the predictions' alignment with physical constraints. Figure 7 shows the absolute difference of the determinant of the predicted rotation matrix and one, the absolute difference of the norm of the predicted points and one, and the test loss, all averaged across numerous points for each angle when using Model 3. First of all, we can see that the desired physical constraint is met perfectly at each training point, which already contributes to smaller overall errors in these regions. Even more important is the effect of the Penalty Method on predictions in areas where training data is sparse. An interesting region to analyse is the one for angles between 0.2 and 2.2, where not a single training data point exists. We can observe both smaller deviation of the determinant from one and smaller test error, meaning that the predictions are not only more realistic, but also closer to the true values. The same behaviour can be seen when using Model 2 in Figure 16 in the appendix. These results together with the overall performance depicted in Figure 6 provide a promising indication that the observed behaviour when applying the introduced methods to Model 3 also generalises to other model architectures. For this reason and due to time limitations, all following experiments are only carried out using Model 3. However, this assumption should be checked in future studies by applying the methods to Model 1 and Model 2.

Increasing physical loss weight

When analysing the version of the Penalty Method where we solve a series of minimisation problems with increasing weights λ for $\mathcal{L}_{DET}$, we first have to discuss finding a good set of parameters. The parameters we need to determine for this method are the following:

- Initial weight $\lambda_0 > 0$,
- Multiplier $\mu > 1$ which determines $\lambda_k = \mu^k \lambda_0$,
- Condition when to stop the current iteration,
- Number of iterations.

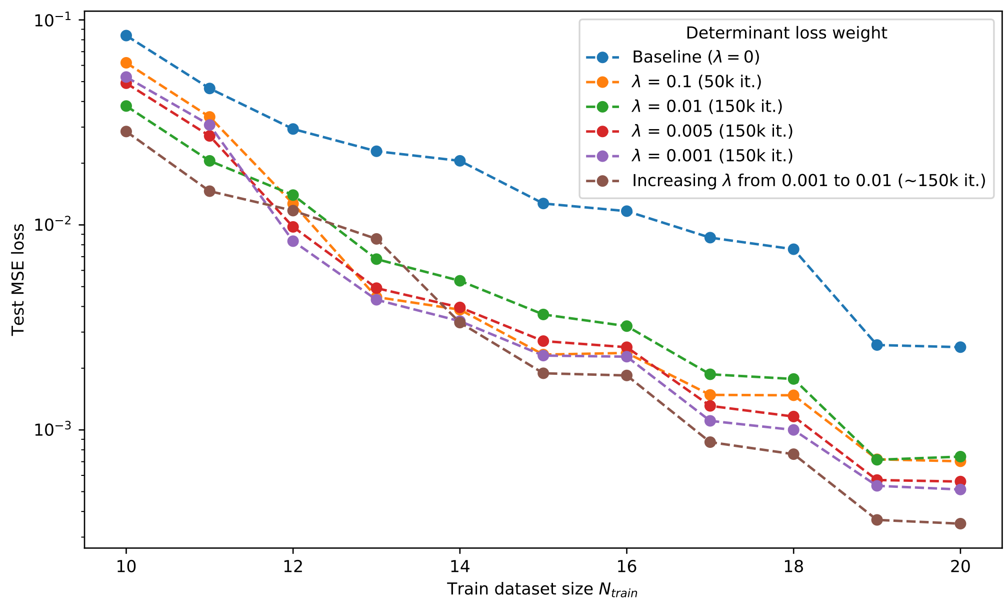


Figure 8: Different weights λ for $\mathcal{L}_{DET}$ compared with dynamically increasing λ, averaged over 10 different training datasets.

In general, we can say that it is difficult to find a successful set of parameters, since we only have an intuition about the initial and last value of λ. We manually tried several parameter sets, of which many led to much worse performance than the results shown above. However, we were able to increase the performance with the following values: We set $\lambda_0 = 10^{-3}$, $\mu = 1.05$, use 50 iterations and stop the k-th iteration once the norm of the gradient is smaller than $0.95^k 10^{-3}$. With this, we get an exponentially increasing series with a last weight value of $\lambda_{50} \approx 10^{-2}$. However, since solving a series of minimisation problems with the chosen parameters involves approximately $150\,000$ iterations, we also compare the results with using the fixed method for $150\,000$ iterations for different values taken by λ during the dynamic training process. The results are illustrated in Figure 8. We can see that the dynamic version outperforms the fixed Penalty Method by a significant margin for most training dataset sizes. For example, for 20 data points, the former achieves an error of 3.5×10^{-4}, whereas the best fixed version only reaches twice the error of 7×10^{-4}. Unfortunately, the improvement is not consistent and for 13 data points, the dynamic version only achieves 8.5×10^{-3}, whereas even the fixed Penalty Method with only $50\,000$ iterations already achieves with a value of 4.3×10^{-3} almost half of this error.

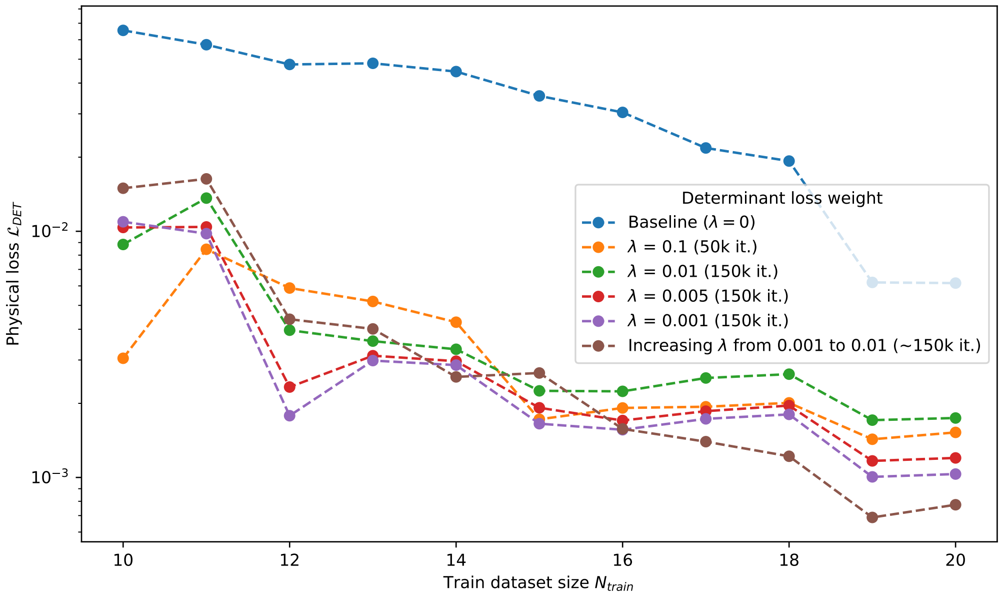


Figure 9: Physical test loss for different weights λ for $\mathcal{L}_{DET}$ compared with the dynamic Penalty Method.

Furthermore, Figure 9 displays the physical loss on test data of different Fixed Penalty Method weights and the Dynamic Penalty Method. When looking at these results, we observe that no single method outperforms all of the other methods in terms of physical feasibility. For example, the Dynamic Penalty Method has the highest physical loss values when trained on 10 and 11 data points, but performs best for 16 to 20 data points. The high deviations between the differences of the compared methods indicate that setting the best weight to achieve physical feasibility heavily depends on the number of training data points. Despite the fact that no method consistently yields the smallest physical constraint violations, it can clearly be seen that for all weights and methods, the determinant loss is significantly smaller than the one of the baseline. This is not surprising, since the latter does not incorporate the knowledge about the physical principles. We further need to mention that these results show that the physical loss is not a reliable indicator for test performance in terms of the MSE-Loss. For example, the Dynamic Penalty Method has the highest Physical Loss for 10 training data points, but at the same time the smallest MSE-Loss value.

As a summary, for the chosen parameters, the dynamic Penalty Method overall achieves a smaller test MSE-Loss than the fixed version, while achieving similar physical feasibility. However, since it requires a multiple of the epochs needed by the fixed method and it is difficult to find a well performing set of parameters, we in general advise to use the simpler method with a fixed physical loss weight λ.

4.3 Augmented Lagrangian Method

We first apply the version which only computes a fixed number of epochs for each minimisation problem, and keep the weight μ of the physical loss $\mathcal{L}_{DET}$ constant. Thus, it differs from the first version of the Penalty Method only in having the additional linear constraint violation term. Note that for ALM, this weight needs to be chosen significantly smaller than for the Penalty Method depending on the constraint violation, since the linear term is much higher relative to the squared constraint violation term for small violation values. Still, it is similarly difficult to find a good set of parameters as it is for the second version of the Penalty Method. We achieved reasonable performance computing 30 iterations with $5\,000$ epochs each and a physical loss weight of $\mu = 0.01$. The Lagrangian multiplier estimates λ are initially set to 0 and updated according to equation (7). Since we have one determinant constraint for every output, the Lagrangian multiplier estimate vector λ contains N_{train} values. The results are depicted in Figure 10 as "ALM fixed". In order to clearly extract the effect of incorporating ALM's core idea of including the linear constraint violation terms, we also include "ALM fixed (no linear)", which uses the same loss function, but omitting the linear terms. The results show that by using ALM with a good set of parameters, we can achieve even smaller error rates than the Penalty Method. For example, for 20 data points, the fixed ALM achieves an error of 3.9×10^{-4}, which is an improvement of 44% compared to the error of 7×10^{-4} of the Penalty Method, and also reduces the error by 24% compared to its version omitting the linear terms, which reaches an error of 5.1×10^{-4}

However, the improvements are not very consistent in the range between 10 and 15 training data points. Further, we have to note that the method also needs $150\,000$ epochs in total compared to the $50\,000$ epochs of the fixed Penalty Method in order to achieve comparable results. An example of the training and test loss after each iteration when applying the fixed ALM is shown in Figure 17 in the appendix. Since the Lagrangian

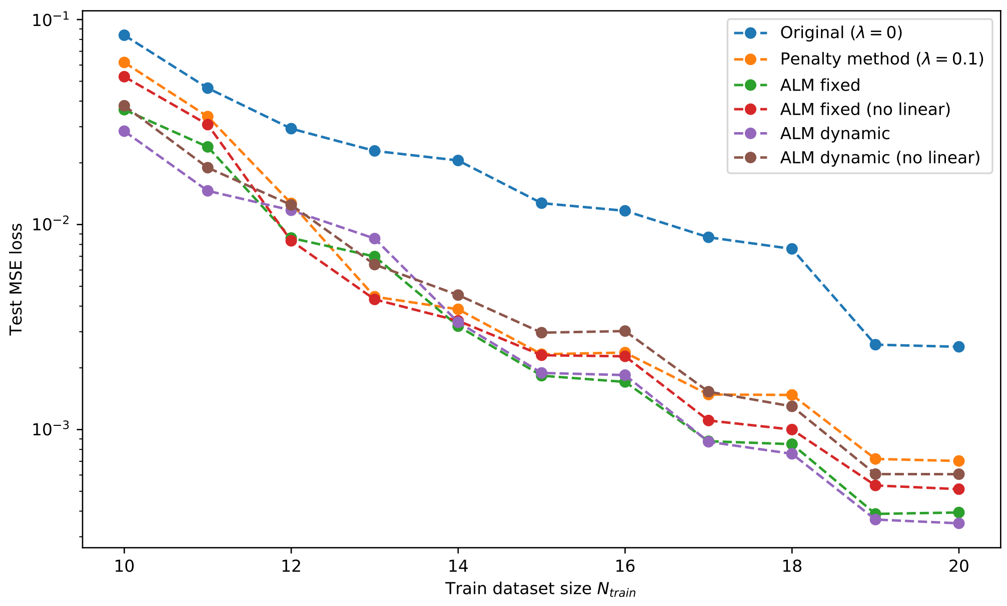


Figure 10: Comparison of ALM, Penalty Method and original training, averaged over 10 different random training datasets.

multiplier estimates can be negative, we included the negative linear loss for the determinant as well. Furthermore, Figure 10 depicts the results of applying the second version of ALM, again including the comparison with the same computation without the linear term in the loss function. As described in section 3.7, we need to determine even more parameters than we do for the first version of ALM or the Penalty Method. This makes it especially difficult to find good parameters, and we need to mention that throughout the process of finding a good parameter set, we have tried many values that led to significantly worse performance than the original model which does not incorporate any physical constraints at all. To get reasonable results, we chose the following parameters: To begin with, we set $\mu_0 = 10^{-3}$ and increased it by a factor of 1.05 after each iteration. By computing 50 iterations, μ increases by approximately one order of magnitude. As before, the Lagrangian multiplier estimates are initially set to 0 and updated according to equation (7). The values for μ_k and λ_k for a single run are depicted in Figure 18 in the appendix. In order to make use of the proposed idea of setting the gradient threshold to a minimum of a fixed value and one which depends on the constraint violation, we set $\gamma_k = 0.95^k \, 10^{-1}$ and $\epsilon_k = 0.95^k \, 10^{-3}$. These values were chosen after analysing the constraint violation values during training and set such that ϵ_k and $\gamma_k ||c(x_k)||$ have approximately the same order of magnitude. The achieved results using these parameters are similar to the fixed ALM results, and again achieve smaller error rates than the corresponding training omitting the linear term. For 20 training data points, it achieves an error of 3.5×10^{-4}, which is close to the 3.9×10^{-4} achieved by the fixed ALM. Overall, this is an improvement of 86% compared to the original model which does not incorpo-

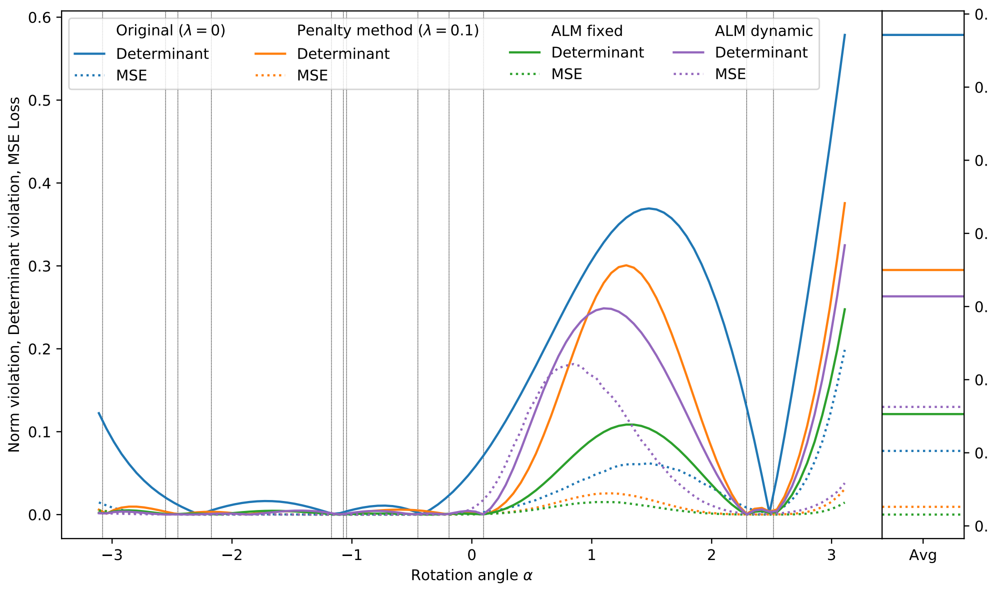


Figure 11: Determinant and norm violations of predictions for $N_{train} = 12$. Gray vertical lines represent the angles of the training points.

rate physical constraints and thus only reaches an error of 2.5×10^{-3}. However, also this version of ALM does not perform consistently better than the Penalty Method. Furthermore, it needs at least $150\,000$ and up to $300\,000$ epochs to achieve these results. Detailed numbers and statistics on the explained experiment are given in the appendix (Table 2 - 4).

To understand the impact of ALM on how well the predictions align with the physical constraints, Figure 11 depicts the absolute difference of the prediction norms and one, the absolute difference of the determinant of the predicted rotation matrix and one, and the corresponding MSE Loss. Note that results vary and Figure 11 only depicts the values for 12 training data points and a single seed used to generate the training dataset.

As we expected, already knowing the statistical performance of the methods, the deviations from the physical constraints are significantly smaller when we apply one of the proposed methods compared to the original training. Similar to the Penalty Method, both versions of the Augmented Lagrangian Method lead to no constraint violations at points where training data is present. While the fixed ALM in this case generalises the physical constraints significantly better than the Penalty Method, the dynamic version achieves only comparable results. Despite, an interesting observation is the high MSE Loss average of the dynamic ALM, which exceeds with a value of 0.033 the error of all other methods significantly. Even the original model achieves an overall average of the MSE Loss of only 0.021. However, this is only due to the performance in the range of angles between 0 and 2, for the extrapolation of angles higher than 2.5, it performs much better than the original model. This is the case even though the determinant in the difficult region is much closer to one than the determinant of the prediction of the original model. This indicates that the ALM focused too much on learning the determinant constraint, since it is able to generalise the determinant constraint only at cost of the performance of the predictions for this specific case. For three dimensions, we were not able to find a set of parameters leading to any improvements compared to the original model.

4.4 Failed experiments

Norm physical loss

As we did for the physical constraint of the determinant being equal to one (13a), we also tried to incorporate the constraint (13b), which says that the norm of the predicted point is supposed to be one. In contrast to what we expected and what we achieved incorporating the determinant constraint, we were not able to improve the performance of our model using the Penalty Method on the violation of the norm constraint. The results for different weights for $\mathcal{L}_{NORM}$ in the training loss for two and three dimensions are shown in Figure 19 in the appendix.

Physical Projection

As explained in section 3.7, we also tested an approach we call Physical Projection. We first applied it to the determinant constraint, thus scaling every predicted matrix with positive determinant according to equation (17). Note that we detach the calculated scaling factor in order to not influence the gradient. Unfortunately, training using this procedure does not converge, an example is shown in Figure 20 in the appendix.

When applying the Physical Projection to the norm constraint, we again detach the calculated norm and only determine training loss and its gradient on the error between the normed prediction and the true point. As we can see in Figure 21 in the appendix, this approach does not improve the performance for either two or three dimensions. When further investigating the reason for this behaviour, we noticed that the norm of the predictions before scaling differs for regions that lack training data between them. The illustration in Figure 22 shows this behaviour well: For $-\pi < \alpha < 0.5$, the determinant of the predicted rotation matrix is only $\frac{1}{10}$ of the predicted determinant for $2 < \alpha < 3$. Even though predictions after the projection align with the physical constraints, it adds complexity to the problem, since the model cannot make use of the rule that the determinant or norm is consistent across all angles.

5 Discussion

To conclude this thesis, we summarise the main results of incorporating physical constraints into deep learning models. First of all, we empirically showed that the knowledge about scientific principles can improve overall performance significantly. In addition, it also leads to higher alignment of the predictions with the physical constraints even in areas where no training data is available. However, since this was only possible using the determinant and not with the norm constraint, we saw that there is no such guarantee for all physical constraints. One might reason that principles that reveal more complicated underlying structures of the process such as the determinant of a rotation matrix lead to higher improvements, since they are otherwise difficult to learn. This is especially true when only few training data is available. We were also able to show that the proposed methods lead to smaller training error in regions where training data is sparse.

When comparing the Penalty Method to the Augmented Lagrangian Method, it is important to note that even though ALM leads to higher performance, there are currently no known methods to the author that estimate hyperparameters well. Since the Penalty Method is easy to understand and to implement, robust to the physical loss weight, and requires fewer epochs, we advise to first apply this method and only search for good ALM hyperparameters once applying the Penalty Method has shown improvements. Furthermore, we were not able to yield improvements using the method of Physical Projections on any constraint.

For future studies, we suggest to check if the achieved improvements shown in this thesis generalise to other model architectures. Furthermore, we propose the idea of training the model on either additional real or simulated unsupervised data using solely the physical loss. This could yield higher physical feasibility of the predictions, which likely improves overall performance, especially in areas where no training data exists. In addition, the positive effects of the introduced methods need to be validated on real world applications. This includes testing their robustness towards commonly occurring problems, such as unbalanced datasets or noise in the training data.

Appendix

Proof that the determinant of the rotation matrix in two dimensions equals one (13a).
Let $\alpha \in [-\pi, \pi]$ be an arbitrary rotation angle. Then we have

$$\begin{aligned} \det(R_{2D}(\alpha)) &= \det \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \\ &= \cos(\alpha)\cos(\alpha) - \sin(\alpha)(-\sin(\alpha)) \\ &= \cos(\alpha)^2 + \sin(\alpha)^2 \\ &= 1 \end{aligned} \tag{18}$$

□

Proof that the determinant of the rotation matrix in three dimensions equals one (13a).
Let $\alpha \in [-\pi, \pi]^2$ be an arbitrary rotation angle. Then we have

$$\begin{aligned} \det(R_{3D}(x, \vec{\alpha})) &= \det(R_y(\alpha_2)R_z(\alpha_1)) \\ &= \det\left(\begin{pmatrix} \cos(\alpha_2) & -\sin(\alpha_2) & 0 \\ \sin(\alpha_2) & \cos(\alpha_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\alpha_1) & 0 & -\sin(\alpha_1) \\ 0 & 1 & 0 \\ \sin(\alpha_1) & 0 & \cos(\alpha_1) \end{pmatrix}\right) \\ &= \det \begin{pmatrix} \cos(\alpha_2) & -\sin(\alpha_2) & 0 \\ \sin(\alpha_2) & \cos(\alpha_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \det \begin{pmatrix} \cos(\alpha_1) & 0 & -\sin(\alpha_1) \\ 0 & 1 & 0 \\ \sin(\alpha_1) & 0 & \cos(\alpha_1) \end{pmatrix} \\ &= (\cos(\alpha_2)^2 + \sin(\alpha_2)^2) \cdot (\cos(\alpha_1)^2 + \sin(\alpha_1)^2) \\ &= 1 \end{aligned} \tag{19}$$

□

Proof for the norm preservation by the rotation in two dimensions (13b).
Let $p = \begin{pmatrix} x \\ y \end{pmatrix}$ be any input vector with $x, y \in \mathbb{R}$ and $\alpha \in [-\pi, \pi]$ an arbitrary rotation angle. Since $|| \cdot || \geq 0$, it is sufficient to show that $||rot_{2D}(p, \vec{\alpha})||_2^2 = ||p||_2^2$.

$$\begin{aligned} ||rot_{2D}(p, \alpha)||_2 &= \left\| \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \right\|_2^2 \\ &= (\cos(\alpha)x - \sin(\alpha)y)^2 + (\sin(\alpha)x + \cos(\alpha)y)^2 \\ &= \cos(\alpha)^2x^2 - 2\cos(\alpha)x\sin(\alpha)y\sin(\alpha)^2y^2 + \sin(\alpha)^2y^2 \\ &\quad + \sin(\alpha)^2x^2 + 2\sin(\alpha)x\cos(\alpha)y + \cos(\alpha)^2y^2 \\ &= (\cos(\alpha)^2 + \sin(\alpha)^2)x^2 + (\sin(\alpha)^2 + \cos(\alpha)^2)y^2 \\ &= x^2 + y^2 \\ &= ||p||_2^2 \end{aligned} \tag{20}$$

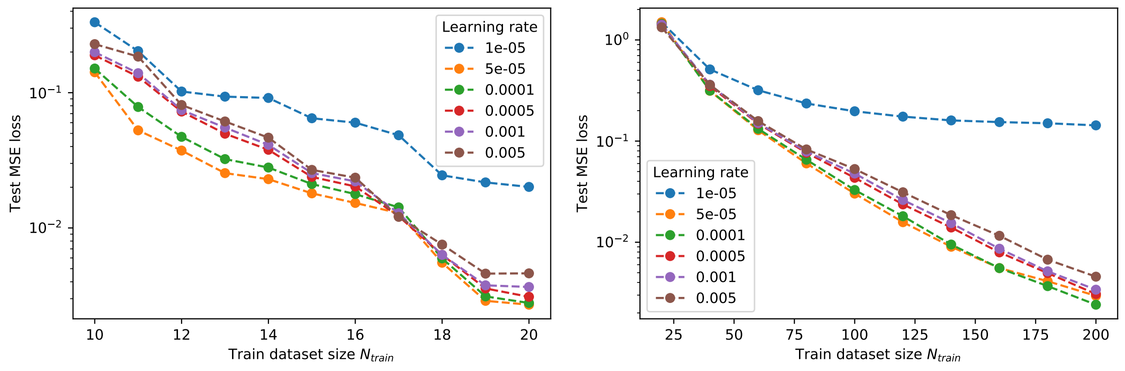
□

Proof for the norm preservation by the rotation in three dimensions (13b).

Let $p = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ be any input vector with $x, y, z \in \mathbb{R}$ and $\alpha \in [-\pi, \pi]^2$ an arbitrary rotation angle. Since $|| \cdot || \geq 0$, it is sufficient to show that $||rot_{3D}(p, \vec{\alpha})||_2^2 = ||p||_2^2$.

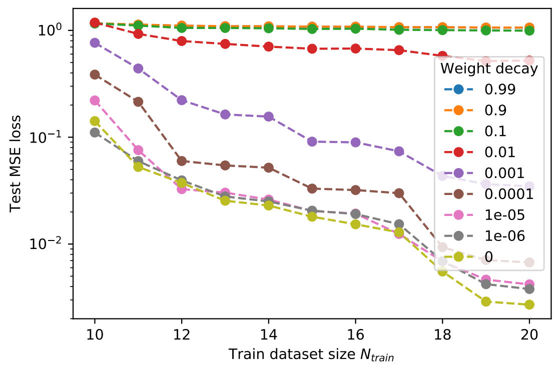
$$
\begin{aligned}
||rot_{3D}(p, \vec{\alpha})||_2^2 &= \|R_y(\alpha_2)R_z(\alpha_1)p\|_2^2 \\
&= \left\| \begin{pmatrix} \cos(\alpha_2) & -\sin(\alpha_2) & 0 \\ \sin(\alpha_2) & \cos(\alpha_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\alpha_1) & 0 & -\sin(\alpha_1) \\ 0 & 1 & 0 \\ \sin(\alpha_1) & 0 & \cos(\alpha_1) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right\|_2^2 \\
&= \left\| \begin{pmatrix} \cos(\alpha_2) & -\sin(\alpha_2) & 0 \\ \sin(\alpha_2) & \cos(\alpha_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\alpha_1)x - \sin(\alpha_1)z \\ y \\ \sin(\alpha_1)x + \cos(\alpha_1)z \end{pmatrix} \right\|_2^2 \\
&= \left\| \begin{pmatrix} \cos(\alpha_2)(\cos(\alpha_1)x - \sin(\alpha_1)z) - \sin(\alpha_2)y \\ \sin(\alpha_2)(\cos(\alpha_1)x - \sin(\alpha_1)z) + \cos(\alpha_2)y \\ \sin(\alpha_1)x + \cos(\alpha_1)z \end{pmatrix} \right\|_2^2 \\
&= (\cos(\alpha_2)^2 + \sin(\alpha_2)^2)(\cos(\alpha_1)x - \sin(\alpha_1)z)^2 + \sin(\alpha_2)^2y^2 + \cos(\alpha_2)^2y^2 \\
&\quad + \sin(\alpha_1)^2x^2 + 2\sin(\alpha_1)x\cos(\alpha_1)z + \cos(\alpha_1)^2z^2 \\
&= \cos(\alpha_1)^2x^2 + \sin(\alpha_1)^2z^2 + y^2 + \sin(\alpha_1)^2x^2 + \cos(\alpha_1)^2z^2 \\
&= x^2 + y^2 + z^2 \\
&= ||p||_2^2
\end{aligned}
\tag{21}
$$

□

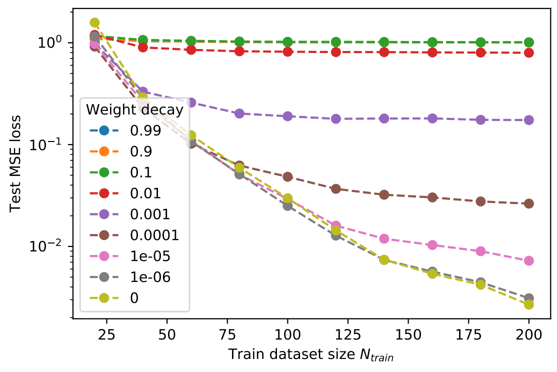


(a) 2 dimensions (b) 3 dimensions

Figure 12: Comparison of learning rates for Model 3



(a) 2 dimensions



(b) 3 dimensions

Figure 13: L2-Regularisation weights on Model 3

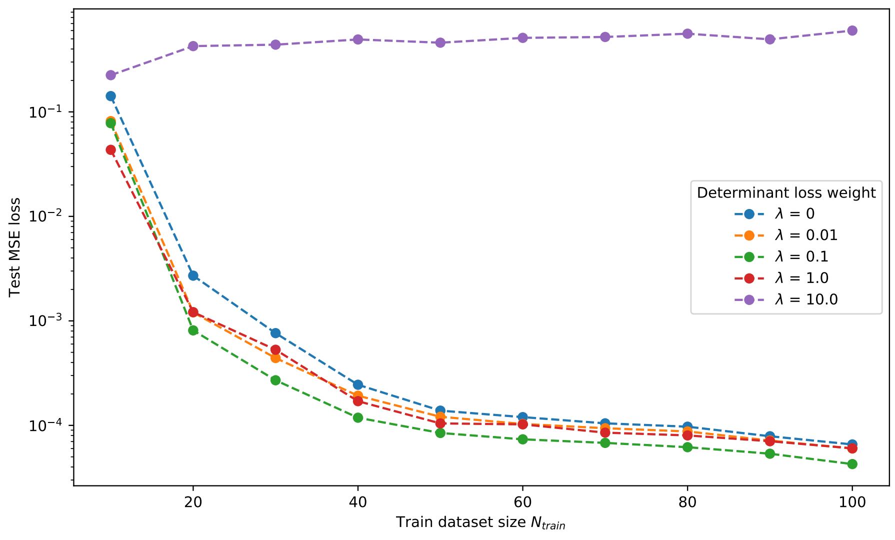


Figure 14: Performance of Model 3 when applying the Fixed Penalty Method with different weights λ for $\mathcal{L}_{DET}$ in 2 dimensions.

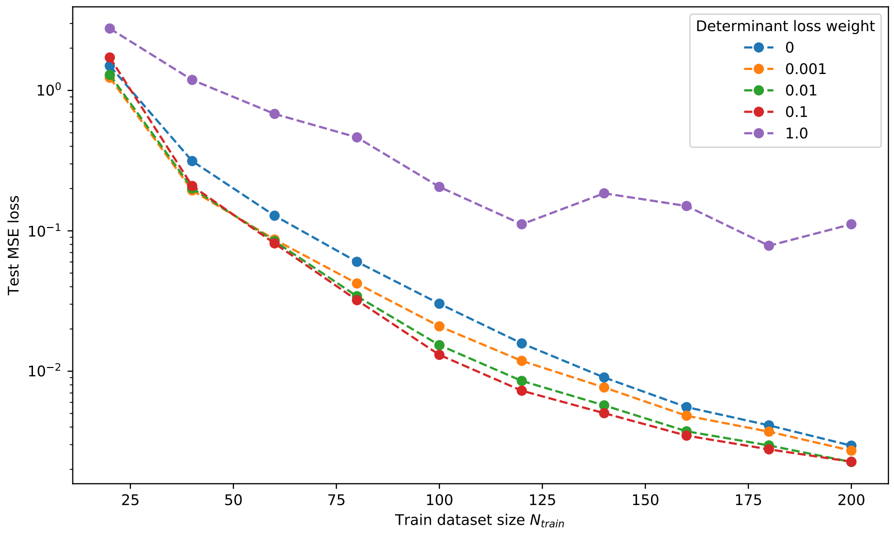


Figure 15: Performance of Model 3 when applying the Fixed Penalty Method with different weights λ for $\mathcal{L}_{DET}$ in 3 dimensions.

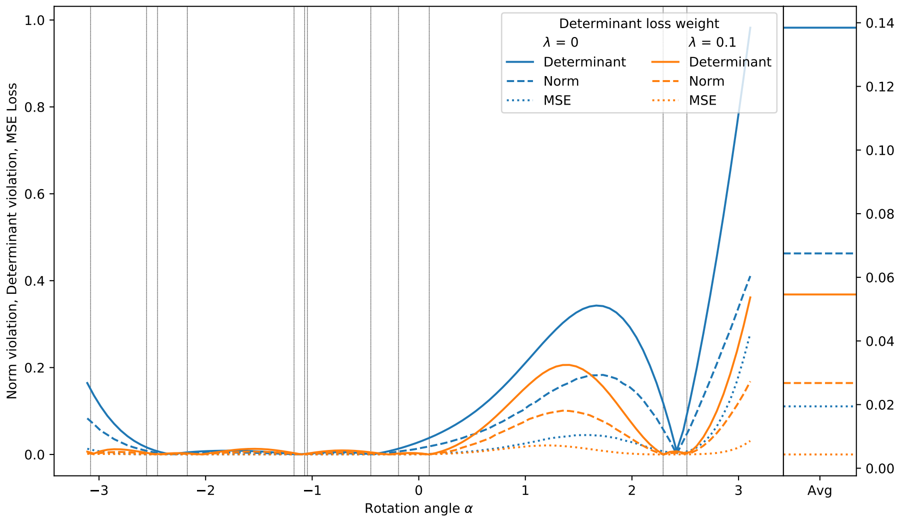


Figure 16: Determinant and norm violations of predictions for $N_{train} = 12$ using Model 2. Gray vertical lines represent the angles of the training points.

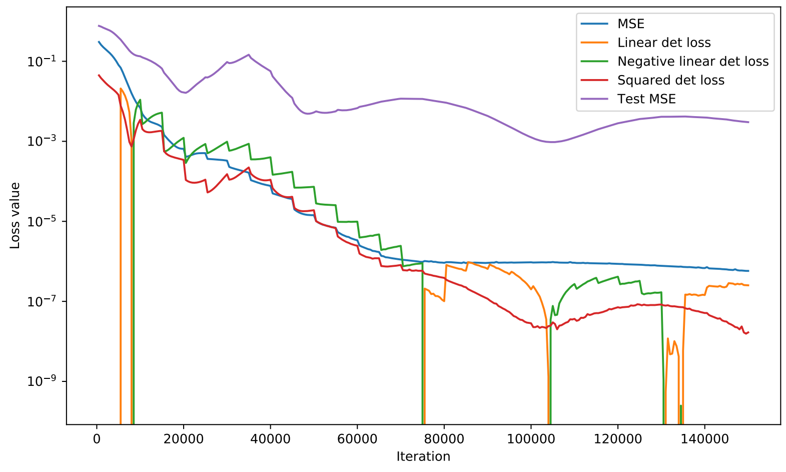


Figure 17: Loss values during training of the fixed ALM in 2D for $N_{train} = 12$ and a single random seed.

	count	mean	std	min	25%	50%	75%	max
Original ($\lambda = 0$)	10.0	0.083873	0.084808	0.017509	0.025126	0.051635	0.102614	0.258126
Penalty method ($\lambda = 0.1$)	10.0	0.061773	0.089446	0.002449	0.006697	0.018631	0.095302	0.280234
ALM fixed	10.0	0.036335	0.041466	0.001796	0.007078	0.014448	0.054447	0.119299
ALM dynamic	10.0	0.028551	0.026700	0.005898	0.009325	0.017179	0.041241	0.079159

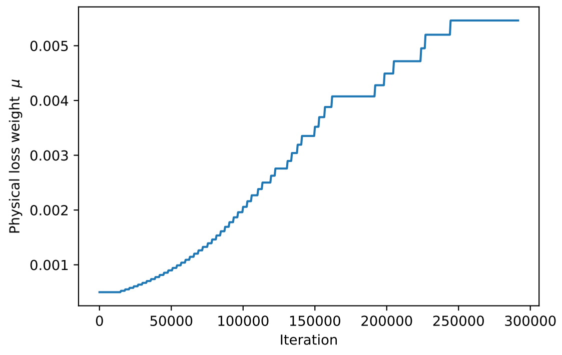
Table 2: Test MSE Loss statistics for 10 random training sets and $N_{train} = 10$

	count	mean	std	min	25%	50%	75%	max
Original ($\lambda = 0$)	10.0	0.012718	0.009706	0.000652	0.006690	0.010347	0.014473	0.032356
Penalty method ($\lambda = 0.1$)	10.0	0.002330	0.001577	0.000174	0.001137	0.002370	0.003468	0.004632
ALM fixed	10.0	0.001832	0.001621	0.000128	0.000656	0.001172	0.002869	0.004497
ALM dynamic	10.0	0.001889	0.001693	0.000094	0.000804	0.001271	0.002364	0.005209

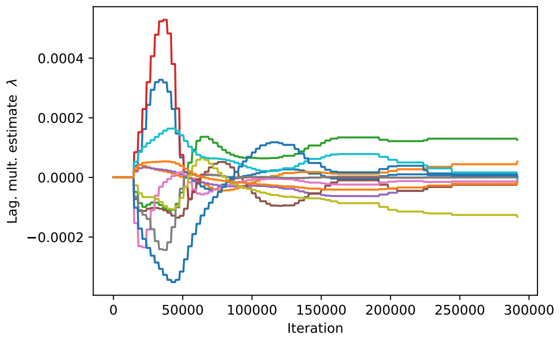
Table 3: Test MSE Loss statistics for 10 random training sets and $N_{train} = 15$

	count	mean	std	min	25%	50%	75%	max
Original ($\lambda = 0$)	10.0	0.002534	0.002381	0.000539	0.000931	0.001674	0.003141	0.007491
Penalty method ($\lambda = 0.1$)	10.0	0.000703	0.000487	0.000154	0.000260	0.000686	0.000999	0.001665
ALM fixed	10.0	0.000394	0.000422	0.000072	0.000140	0.000174	0.000511	0.001419
ALM dynamic	10.0	0.000349	0.000332	0.000016	0.000072	0.000211	0.000564	0.000922

Table 4: Test MSE Loss statistics for 10 random training sets and $N_{train} = 20$

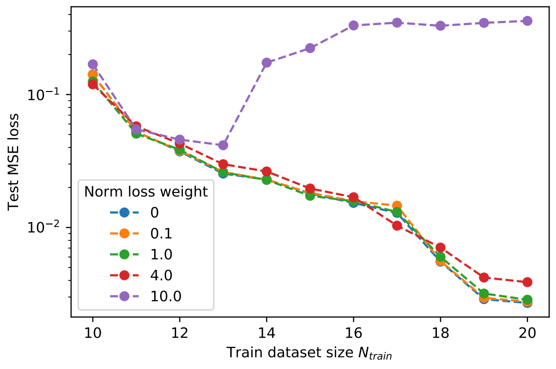


(a) Physical loss weight μ

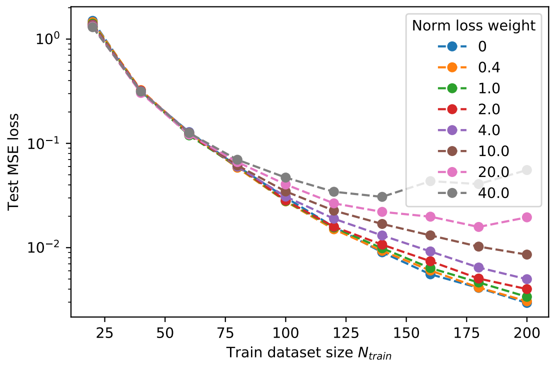


(b) Lagrangian multiplier estimates λ

Figure 18: Values for the physical loss weight μ and the Lagrangian multiplier estimates λ when training using the dynamic ALM with $N_{train} = 12$.



(a) 2 dimensions



(b) 3 dimensions

Figure 19: Penalty Method on the Norm constraint

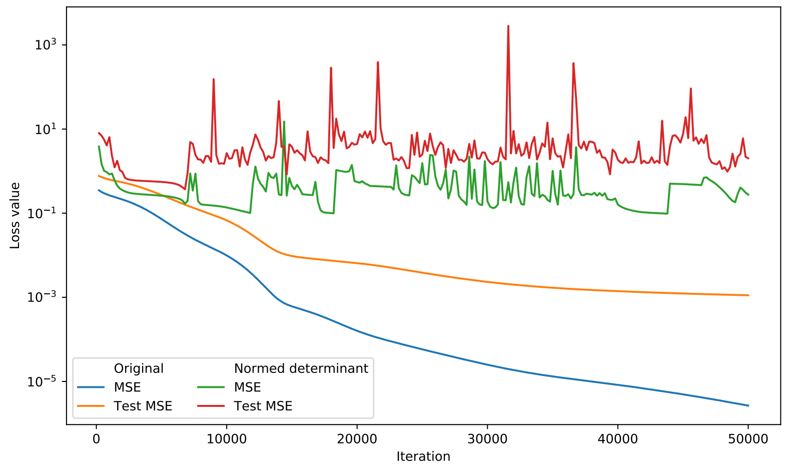
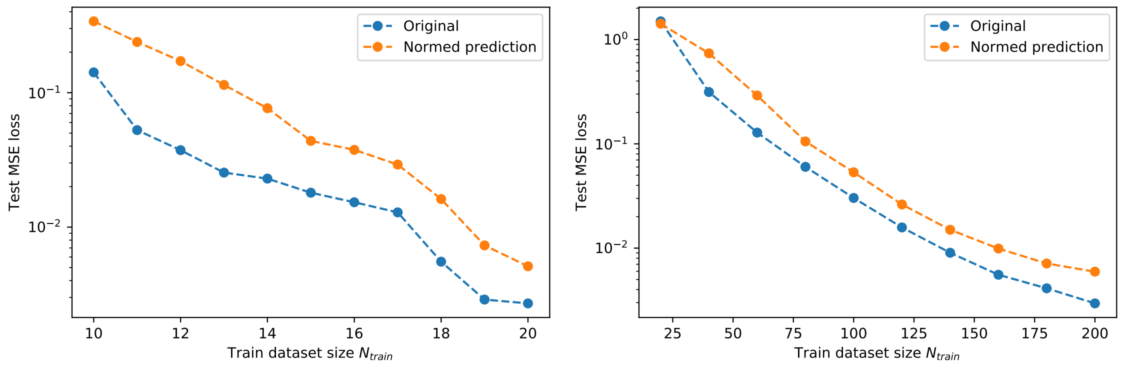


Figure 20: Training for 2 dimensions and $N_{train} = 20$ using the original training and the Physical projection on the determinant with the same training dataset.



(a) 2 dimensions

(b) 3 dimensions

Figure 21: Physical projection on the norm of the prediction

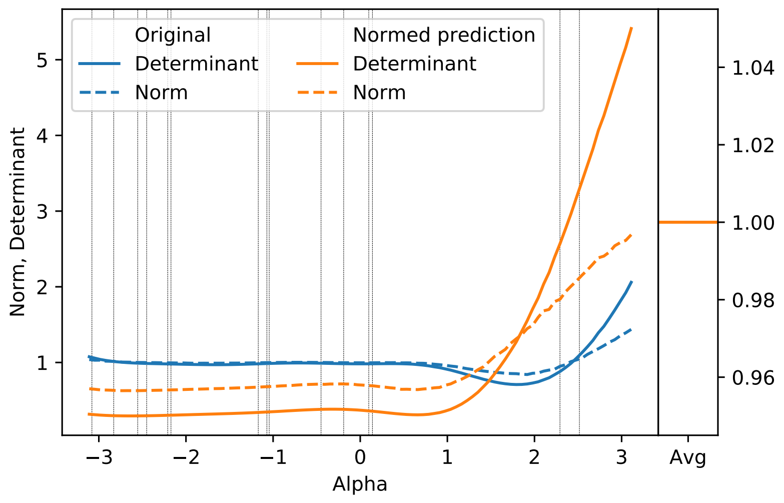


Figure 22: Analysis of the predictions of a model trained using Physical Projection on the norm compared to the original model. All determinants and norms are scaled such that their mean is equal to one.

Bibliography

[1] Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19), 2017.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*, pages 215–272. Cambridge University Press, New York, NY, USA, 2004.

[3] Yuxin Chen. Lecture notes in ele 538b: Large-scale optimization for data science, Spring 2018.

[4] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[6] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando A. Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving. *CoRR*, abs/1504.01716, 2015.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[9] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[10] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[11] Jack Yurkiewicz. Constrained optimization and lagrange multiplier methods, by d. p. bertsekas, academic press, new york, 1982, 395 pp. price: $65.00. *Networks*, 15:138–140, 1985.