Using Autoencoders for reduced order modeling of the BGK model

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
im Fach Physikalische Ingenieurswissenschaften

Technische Universität Berlin
Fakultät Verkehrs- und Maschinensysteme V
Institut für Numerische Fluiddynamik

eingereicht von: *Zachary Schellin*
geboren am: *11.02.1991, Berlin*

Gutachter: *Prof. Dr. Julius Reiss*
Dr. Stefan Schaefer

eingereicht am: *19. Februar 2021*

Abstract

Abstract here

Zusammenfassung

german abstrac here

Contents

Contents **i**

1 Introduction **1**
1.1 Objective of this thesis . . . 1
1.2 Thesis outline . . . 1
1.3 State of the art . . . 2

2 The BGK Model **3**
2.1 Discretization, conservative properties and equilibirum . . . 3
2.2 Sod's schock tube as a test case for the BGK model . . . 4

3 Dimensionality reduction algorithms **6**
3.1 Proper orthogonal decomposition (POD) . . . 6
3.2 Autoencoders . . . 6
3.2.1 Autoencoders . . . 9

4 Reduced Order Model **10**
4.1 Offline Phase . . . 11
4.1.1 Full order BGK model . . . 11
4.1.2 Reduced Basis by POD and Autoencoder . . . 11
4.1.3 Online Phase . . . 13
4.1.4 Reduced Order Model . . . 13

5 Results **14**
5.0.1 Hydrodynamic Regime . . . 19
5.0.2 Discussion and Outlook . . . 22

A Hyperparameters for the Fully Connected Autoencoder **24**

B Hyperparameters for the Convolutional Autoencoder **39**
B.0.1 Appendix B . . . 39

Bibliography **42**

1 Introduction

The Bhatnagar-Gross-Krook equation (BGK) is a kinetic collision model of ionized and neutral gases valid for rarefied as well as other pressure regimes [1]. Generating data of such a flow field is essential for various industry and scientific applications[**REF**]. With the intention to reduce time and cost during the data generating process, experiments were substituted with computational fluid dynamics (CFD) computations. Consequently reduced-order models (ROMs) coupled to aforementioned computations were introduced to further the reduction of time and cost. The thriving field of artificial intelligence operates in model order reduction for data visualization/analysis since the 80's (Quelle?) and has now surfaced in fluid mechanics. This thesis will cover the use of artificial intelligence for model order reduction in fluid mechanics.

1.1 Objective of this thesis

Due to the non-linearity of transport problems in particular shock fronts, the construction of a robust ROM for those cases poses several challenges.

1.2 Thesis outline

1. What is the BGK Model
2. What is the SOD and BGK in SOD
3. What is deep learning what are autoencoders
4. hyperparameters for autoencoders
5. What is a reduced order model
6. offline phase
7. my data in 1d bgk in sod shock tube FOM Data
8. what is pod and Rb by POD
9. RB by autoencoder -online phase
10. what is my ROM
11. results - results by ROM for FC and for CONV

12. variation of intrinsic variables

13. Comparion to POD intrinsic variables number and quality

1.3 State of the art

State of the art model reduction of dynamical systems can be done via proper orthogonal decomposition (POD) which is an algorithm feeding on the idea of singular value decomposition (SVD)[2][3]. POD captures a low-rank representation on a linear manifold. So called POD modes, derived from SVD, describe the principle components of a problem which can be coupled within a Galerkin framework to produce an approximation of a lower dimension r.

$$f(x) \approx \tilde{f}(x) \quad \text{with} \tilde{f} = \sum_{k=1}^{r} a_k \psi_k(x) \qquad \text{where} \qquad \psi_k \quad \text{are orthonormal functions.} \tag{1.1}$$

Bernard et al. use POD-Galerkin with an additional population of their snapshot database via optimal transport for the proposed BGK equation, bisecting computational run time (cost) in conjunction with an approximation error of $\sim$ 1 % in [4]. Artificial intelligence in the form of autoencoders replacing the POD within a Galerkin framework is evaluated against the POD performance by Kookjin et al. for advection-dominated problems[5] resulting in sub 0.1% errors. An additional time inter- and extrapolation is evaluated. Using machine learning/ deep learning for reduced order modeling in CFD is a novel approach although "the idea of autoencoders has been part of the historical landscape of neural networks for decades"[6, p.493]. Autoencoders, or more precisely learning internal representations by the delta rule (backpropagation) and the use of hidden units in a feed forward neural network architecture, premiered by Rumelhart et al. (1986) [7]. Through so called hierarchical training Ballard et al.(1987) introduce a strategy to train auto autoassociative networks (nowadays referred to as autoencoders), in a reasonable time promoting further development despite computational limitations [8]. The so called bottleneck of autoencoders yields a non-smooth and entangled representation thus beeing uninterpretable by practitioners[9] leading to developements in this field. Rifai et al. introduce the contractive autoencoder (CAE) for classification tasks (2011), with the aim to extract robust features which are insensitive to input variations orthogonal to the low-dimensional non-linear manifold by adding a penalty on the frobenius norm of the instrinsic variables with respect to the input, surpassing other classification algorithms [9]. Subsequent development emerges with the manifold tangent classifier (MTC) [10]. A local chart for each datapoint is obtained hence characterizing the manifold which in turn improves classification performance. On that basis a generative process for the CAE is developed. Through movements along the manifold with directions defined by the Jacobian of the bottleneck layer with respect to the input $\vec{x}_m = JJ^T$, sampling is realized [11].... Proper orthogonal decomposition (POD) and it's numerous variants like shifted-POD[**?**], POD-Galerkin[**?**], POD+I [**?**] to name only a few of them, try to solve this problem by.....

2 The BGK Model

This section covers the kinetic gas model, the BGK model, on which a model order reduction will be performed in the following. In addition the SOD-shocktube, on which the BGK model will be tested is discussed.

2.1 Discretization, conservative properties and equilibirum

The BGK model was introduced by, and named after, physicists Prabhu L. Bhatnagar, Eugene P. Gross and Max Krook in 1954 [1]. It is an approximation of the standard Boltzmann transport equation. More precisely the r.h.s of the Boltzmann equation is approximated by the BGK operator [12]

$$\partial_t f + v\partial_x f = \frac{1}{\tau}(M_f - f). \tag{2.1}$$

It consists of the relaxation time $\tau(x,t)$, the Maxwellian distribution M_f and $f(x,v,t)$ the probability of a gas particle having a microscopic velocity v in phase space (x,v,t). The left side of the BGK model is a transport equation for $f(x,v,t)$ with transport velocity v. In the maxwellian distribution $n(x,t)$ is the number of particles in space and time, R is the specific gas constant, $T^0(x,t)$ is the temperature and $u(x,t)$ is the macroscopic velocity

$$M_f = \frac{n(x,t)}{\sqrt{2\pi RT^0(x,t)}} \exp(-\frac{(v-u(x,t))^2}{2RT^0(x,t)}). \tag{2.2}$$

Note that in this thesis the BGK model is discussed in one dimension. Hence the BGK model needs to be evaluated for three independent variables x, v and t as seen above.
Now what makes the BGK model especially attractive for model order reduction? The fruitfullness of performing a model order reduction on the BGK model becomes clearer when looking at it's space and velocity discretization

$$\partial_t f_{j,k} = -(v_k)_1 D_x f|_{j,k}(t) + \frac{1}{\tau}(M_{f_{j,k}}(t) - f_{j,k}(t)). \tag{2.3}$$

Here a uniform grid is considered with $x_j = j\Delta x$, $j \in \mathbb{Z}$, $v_k = k\Delta v$, $k \in \mathbb{Z}$ and $t^n = n\Delta t$, $n \in \mathbb{N}$ on which $f_{j,k} = f(x_j, v_K, t)$ and $M_{f_{j,k}} = M_f(x_j, v_k, t)$ are evaluated at point (x_j, v_k) in a time instance t. For brevity $D_x f|_{j,k}$ is the discrete space derivative at (x_j, v_k) [12]. Now the PDE in eq. (2.1) is broken down into a system of ODE's in time, for which every ODE is a lineaar advection equation with constant scalar speed v_k and a source term.
To continue let's consider K to be the number of gridpoints in velocity and J to be the number

of grid points in space. Then a total of KJ first order differential equations need to be evaluated in 1D. Obviously in three dimensions the system of ODE's inflates up to K^3N^3 first order differential equations eq. (2.3). This in turn drives the evaluation of the BGK model at the edge of intractability for dense meshes in 3D and all together stimulate the want for a reduced order model.

In order to evaluate M_f and therefore eq. (2.3) the macroscopic velocity $u(x,t)$ is required. Other macroscopic quantities of the gas flow, namely the density ρ, the momentum ρu and the energy E are the expected values or moments of f in velocity space and can be calculated with

$$\rho(x,t) = m \int f \, \mathrm{d}v, \tag{2.4}$$

$$\rho(x,t)u(x,t) = m \int vf \, \mathrm{d}v, \tag{2.5}$$

$$E(x,t) = m \int \frac{1}{2} v^2 f \, \mathrm{d}v, \tag{2.6}$$

as seen in [12]. Here f is integrated over velocity space and multiplied by $m\Phi$ with $\Phi = [1, v, \frac{1}{2}v^2]$, the collision invariants and m the mass of the particles.

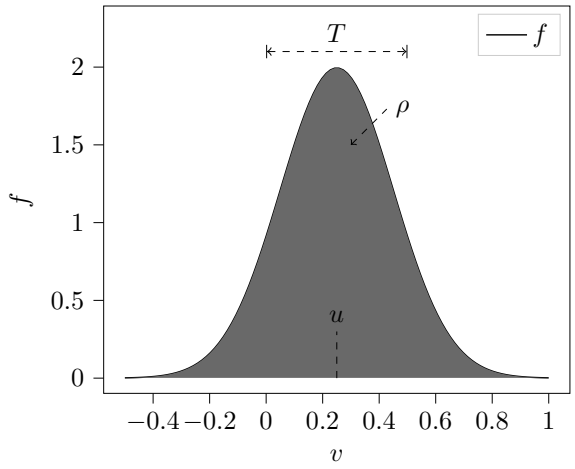


Figure 2.1: Illustration of the linkage between the macroscopic quantities of the gas flow and f the distribution function f.

Displayed in fig. 2.1 is a demonstrative example of how the distribution function $f(v)$ gives the values for the macroscopic quantities. The distribution is centered around the macroscopic velocity u, the mean velocity of the distribution f is the temperature T, integrating $f(v)$ over velocity space one obtains the density ρ.

Evidently the system in eq. (2.1) is in equilibrium when $f = M_f$. Now multiplying the equilibrium solution of eq. (2.1) by $m\Phi(v)$ and integrating in velocity space, one finds the Euler system of classical gas dynamics using the equation of state of the gas [12]

$$\partial_t \rho + \partial_x(\rho u) = 0 \tag{2.7}$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + p) = 0 \tag{2.8}$$

$$\partial_t E + \partial_x(u(E+p)) = 0. \tag{2.9}$$

Note that the Boltzmann transport equation the conservative properties of the microscopic collisions lead to a global conservation of mass, momentum and energy. Hence the conservation of mass, momentum and energy is found in the BGk model as well [12].

To continue ..... rarefaction KN SOD

2.2 Sod's schock tube as a test case for the BGK model

- intrinsic code variables for hydro an rare add pls

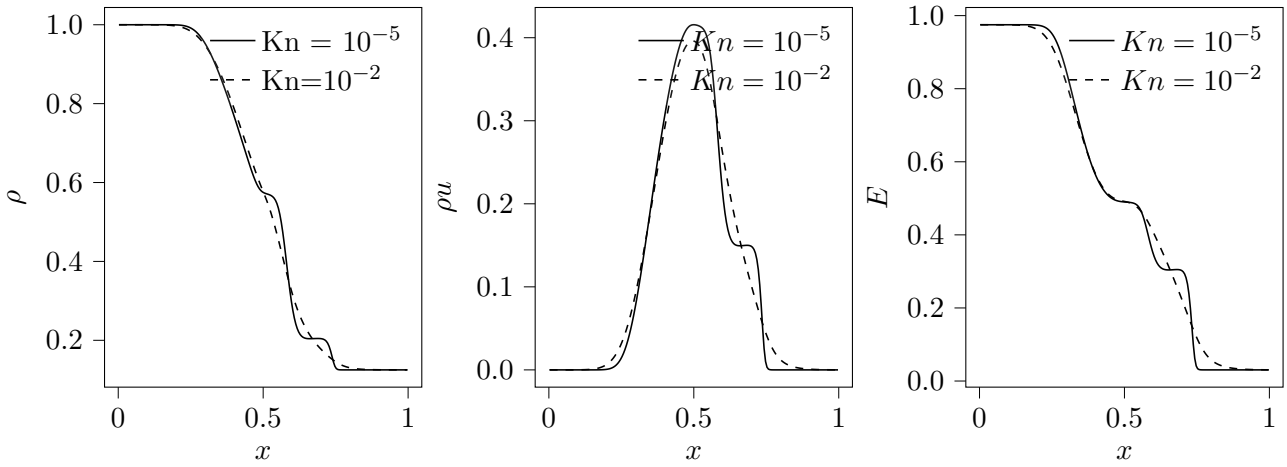


Figure 2.2: Macroscopic quantities rho, ρu and E in the SOD schock tube at the last time stamp. The quantites are displayed for the hydrodynamic regime in - - lines and for the rarefied regime in - lines.

3 Dimensionality reduction algorithms

3.1 Proper orthogonal decomposition (POD)

3.2 Autoencoders

In this section deep learning with the focus on fully connected autoencoders and convolutional autoencoders will be introduced. Starting of with addressing important terminology whilst presenting the concept of autoencoders and continuing with the introduction of the fully connected and convolutional forwardpass , this section closes with ADAM [**?**] an update to the backpropagation algorithm as well as important training methods.
The term deep learning situated around the much broader field of artificial intelligence stems from the use of deep feed forward networks also called multi layer perceptrons (MLP) or feed forward neural networks. Deep recurrent neural networks (RNN) are also used in this field but won't be covered in this thesis. In contrast to RNNs, information flows forward through these networks, which explains the name feed foward. In the following I will use the abbreviation MLP when talking about the aforementioned algorithm. Network refers to the typical composition of many different functions.
The task of any MLP is to approximate a function $f^\star(\mathbf{x};\boldsymbol{\Theta}) \approx f(\mathbf{x})$ through learning the values of the parameters $\boldsymbol{\Theta}$. As mentioned before $f^\star$ is a composition of functions eg. eq. (3.1).

$$f^\star(\mathbf{x};\boldsymbol{\Theta}) = f^3(f^2(f^1(\mathbf{x};\boldsymbol{\Theta}^1),\boldsymbol{\Theta}^2),\boldsymbol{\Theta}^3) \tag{3.1}$$

In eq. (3.1) each function f^i is called layer. In this example f^1 is called the input layer, f^3 the output layer and f^2 a hidden layer. Hence a layer is a vector-to-vector function. In this context a unit describes the corresponding vector-to-scalar functions of one layer. The width of a layer is referred to as the dimension of the vector valued input. The depth of the network describes the number of composed functions. In autoencoders the dimensions of input and output layer are identical.
Autoencoders are a special kind of neural network, that have a central hidden layer, that outputs a code c which should contain useful features of the input x while usually being of a lower dimension. Hereafter the code will be addressed as intrinsic variables highlighting the property of containing useful features of the input x. Autoencoders can be split in two parts, the encoder $h(x) = c$ which compresses the input and outputs the intrinsic variables c and the decoder $g(h) = \hat{x}$ which reconstructs the input from the intrinsic variables to output $\hat{x}$. The goal of autoencoders conflicts with the training objective. The former is to produce a code that describes the intrinsic features of the input, while the latter is to minimize the difference between x and $\hat{x}$. Therefore autoencoders need to be restrained from learning the identity function perfectly which in turn should drive the

model to choose which instance to copy.
Obviously deep learning emphasizes the focus on the depth of a model. This is because linear models with just one layer can only approximate linear functions. Adding a non-linear activation function to the output of the proposed model wouldn't be sufficient in modeling any nonlinear behavior of the function. However the universal approximation theorem [13] states that MLPs with at least one hidden layer and any non-linear activation function can approximate any function given that enough hidden layers can be provided. In conclusion, MLPs are universal approximators [6].
There are several types of layers, that can be used in MLPs. For this thesis it is sufficient to treat so called *fully connected layers* and *convolutional layers*. Fully connected layers are called `Linear`[**?**] in `PyTorch`[**?**] because they compute a linear transformation of the input eq. (3.2), where x is the input vector, A is the weight matrix and b is a bias vector. This is the forward pass of a linear layer. The learnable parameters Θ are in this case the values in A and b. For a linear layer which takes a vector of size i as input and outputs a vector of size o there are $l = i \times o + o$ learnable parameters.

$$y = xA^T + b \tag{3.2}$$

Continuing with the forward pass in convolutional layers, which are called `Conv2D`[**?**] in `Pytorch`. Convolutional layers are usually applied when the input data has a known grid-like topology [6].While for fully connected layers, the input size is fixed, convolutional layers can be applied to inputs of various sizes. Furthermore, they are sparse by construction and share parameters making them equivariant [6]. Even tough the name implies the use of the convolutional operation in eq. (3.3), `PyTorch` and many other neural network libraries instead use the cross correlation, which is an operation closely related to a convolution[6][**?**]. The convolution in 3.3 operates on two functions x and w, where the latter is a weighting function of the former which makes s the weighted average of the input x. In eq. (3.4) x is represented by $I(m, n)$, and w by $K(m, n)$ respectively. Note that while eq. (3.3) is scalar valued and eq. (3.4) is two dimensional, only for illustrative reasons.

$$s(t) = (x * w)(t) = \int x(a)w(t - a)\, da \tag{3.3}$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{3.4}$$

One drawback in implementing the discrete convolution eq. (3.4) is that there can be invalid values for m and n. This can be solved by exploiting the commutative property of the convolution, which results for the discrete convolution in a flipped kernel relative to the input, eq. (3.5). Without the need of flipping the kernel which is not always possible, i.e. exploiting the commutative property, cross correlation is adopted eq. (3.6).

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{3.5}$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \tag{3.6}$$

The given equations illustrate the movement of the kernel over a two dimensional input, but leaving out two basic accompanying designs. First is the so called stride of the kernel, which results in a increased down sampling with increased stride size. Considering strides of the kernel along one

dimension of the input results in a shrinkage of that dimension by

$$o = \frac{i - k}{s} + 1. \tag{3.7}$$

Here o, i, s and k are the output, input, stride and kernel size of one dimension respectively [14]. Second are the so called channels, which allow convolutional layers to extract a different feature for every channel from the same input. For a two dimensional input like images this could be different features for the same location on the image, like edges and color in the RGB color space [6]. Adding strides and kernel to eq. (3.6) yields

$$\mathsf{S}_{i,j,k} = c(\mathsf{K}, \mathsf{I}, s)_{i,j,k} \sum_{l,m,n} \left[\mathsf{I}_{l,(j-1)\times s+m,(k-1)\times s+n} \mathsf{K}_{i,l,m,n} \right]. \tag{3.8}$$

The kernel K is a four dimensional tensor with i indexing into the output channels of S, l indexing into the input channels of I, m and n indexing into the rows and colums. The input I and output S are three dimensional tensors with j and k indexing into the rows and columns. Note that in `PyTorch` eq. (3.8) is a so called valid cross correlation (valid convolution) [**?**] meaning the kernel will only move over input units for which all m and n are inside the rows and columns of the input. Zero padding the input can prevent the kernel from omitting corners in that case.
For autoencoders the necessity to perform a transposition of the applied layers arises, which is straight forward for fully connected layers. Convolutional layers with strides greater than unity on the other hand the transposition needs the kernel to be padded with zeros to realize an upsampling of the input data [14]. Note that the padding of the kernel with zeros is only used to illustrate how the upsampling works. In eq. (3.9) taken from [6] multiplications with zero are omitted. The size of one dimension during upsampling can be calculated with the transposition of eq. (3.7).

$$t(\mathsf{K}, \mathsf{H}, s)_{i,j,k} = \sum_{\substack{l,m \\ s.t \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ s.t \\ (n-1)\times s+p=k}} \sum_{q} \mathsf{K}_{q,i,m,p} \mathsf{H}_{q,l,n} \tag{3.9}$$

Forward-propagation is then referred to as the compositional evaluation of each layer. For the example in eq. (3.1) the outputs of each layer would then be: $f^1(\mathbf{x}, \mathbf{\Theta}^1) = \mathbf{p}$, $f^2(\mathbf{p}, \mathbf{\Theta}^2) = \mathbf{q}$ and $f^3(\mathbf{q}, \mathbf{\Theta}^3) = \hat{\mathbf{y}}$.
Subsequently the cost function $J(\Theta)$, which will be discussed later on, can be computed. Afterwards back-propagation returns the gradients w.r.t. the layer parameters Θ to finally compute updated parameters $\mathbf{\Theta}$. The name back-propagation refers to the use of the chain rule of calculus to obtain the gradients of each layer. Assuming again the example in eq. (3.1) back-propagation would be equations eq. (3.12) to eq. (3.10):

$$\frac{\partial J}{\partial \mathbf{\Theta}^3} = \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}}{\partial \Theta^3} \tag{3.10}$$

$$\frac{\partial J}{\partial \Theta^2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial q} \frac{\partial q}{\partial \Theta^2} \tag{3.11}$$

$$\frac{\partial J}{\partial \mathbf{\Theta}^1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial q} \frac{\partial q}{\partial p} \frac{\partial p}{\partial \Theta^1} \tag{3.12}$$

While the term back-propagation is solely used for the method to compute the gradients for each layer in a backward fashion, meaning form the last layer to the first, the update of the parameters is done in an optimization step. -L2-error as perfomance metrics

3.2.1 Autoencoders

Autoencoders have many hyperparameters determining their capability for compression and subsequent reconstruction. These parameters include : *depth*, *width of layers*, *activation functions*, *batch-size*, *learning rate*, *number of filters*, *stride width*, *kernel size*. Their finding is discussed in this section, for both **H** and **R**.
For the fully connected autoencoder the order of determining the hyperparameters is as follows: Depth → Hidden Units → Batch Size → Activation Fuctions. For the convolutional autoencoder the order of determining the hyperparameters is as follows: Depth → Channels → Batch Size → Activation Functions. Both models are evaluated on the course of the training. Figures showing the training process for all experiments are provided in the appendix. After training the L2-Error**??** is evaluated on the unshuffled, complete dataset as described in section 4.1.1.
Important to mention is that the difficulty of finding the right set of hyperparameters for MLPs led to an intensive search prior to the creation of this contribution. The difficulty lies in the fact that there is little systematic knowledge about how the hyperparameters interact in the model and answer to a variety of input data. Goodfellow et al. point out that with a combination of intuition, certain methods and first of all experience practitioners find hyperparameters that work well [6]. As for this thesis a list of the prior search is provided in appendix B.0.1. This list does not claim to cover the complete search and can by no means give enough insight to enable reproducability by another person.
For this reason an insight into the methods employed to find the set of hyperparameters used in this contribution are described in the following.

The convolutional autoencoder architecture 1.1 is a composition of six convolutional and three fully conected layers, eq. (3.13).

$$y_p = f_C^9(f_C^8(f_C^7(f_F^6(f_F^5(f_F^4(f_C^3(f_C^2(f_C^1(y_0)))))))))) \tag{3.13}$$

4 Reduced Order Model

In this section model order reduction (ROM) will be introduced and two algorithms for obtaining a reduced basis (RB) are discussed. The proper orthogonal decomposition (POD) and autoencoders. In addition a reduced order model (ROM) based on the method of characteristics[15] is evaluated. Model order reduction is a technique used for reducing the computational cost, which is computational resources as memory and computation power. Partial differential equations (PDEs) once discretized become a system of high dimensional ordinary differential equations (ODEs) as shown in **??**. Here model order reduction exploits the idea that every high dimensional dynamical-state space$f(\mathbf{x}, \mu) \in \mathcal{D}$ can be described by a state-space or manifold of lower dimension $\tilde{f}(\mu) \in \mathcal{E}$

$$f \approx \tilde{f} \qquad \text{with} \qquad \mathcal{D} \ll \mathcal{E}. \tag{4.1}$$

Reduced order modeling is partitioned into two successive phases called the *offline* - and the *online phase*. During the offline phase data or *snapshots* of a dynamical-system is generated through experiments or simulations of the full order model (FOM). The so called *snapshots* $U = u(t1), ..., u(t_n)$ are created once, each representing one moment in time of the dynamical system. Next a mapping g is constructed such that $\tilde{u} = g(\tilde{u})$, for which $u(t_i) \approx u(t_i)$. During the online phase the reduced order model is evaluated and the error is estimated by eg. $||u(t) - \tilde{u}(t)$. Therefore the online phase may be described as stage of independence from the full order model. To continue the definition of the *intrinsic solution manifold dimensionality* by *Carlberg et al.* [5] is needed: Assuming the initial value problem

$$\mathbf{r}^n(\mathbf{x}^n; \mu) = 0, \qquad n = 1, ..., N_t \tag{4.2}$$

has a unique solution for each parameter instance $\mu \in \mathcal{D}$, the intrinsic dimensionality of the solution manifold $\{\mathbf{x}(t, \mu) | t \in [0, T], \mu \in \mathcal{D}\}$ is (at most) $p^\star = n_\mu + 1$, as a mapping $(t, \mu) \mapsto \mathbf{x}$ is unique in this case. This provides a practical lower bound on the dimension of a nonlinear trial manifold for exactly representing the dynamical-system state. In summary the intrinsic solution manifold dimensionality, in the following referred to as the number of intrinsic variables, are in number as much as there are parameters that can describe the whole dynamical-system state plus one, the time t. As for the full order BGK equation **??** the intrinsic variables could be three for the macroscopic velocity $U(\mathbf{x}, t)$, three for the microscopic velocities $\xi$of the collision operator $Q(f, f)$, three for T,ρ and ν plus one equaling to a total of ten intrinsic variables for the 3D case. In the following subsection the available data for this thesis is introduced along with a proposal for the number of intrinsinc variables.
The BGK equation is valid for gases in the hydrodynamic regime as well as for rarefied gases. As described in **??** depending on the equilibrium of the BGK equation, the solution transitions between a pure Boltzmann - and a Maxwellian distribution. The former is known to be well approximated by linear methods as the SVD, while the latter poses problems due to the non-linear behavior.

4.1 Offline Phase

4.1.1 Full order BGK model

The number of intrinsic variables can therefore be determined for the 1D case of the BGK equation. The parameters μdescribing the gas flow in the hydrodynamic regime has therefore one macroscopic velocity $U(x)$, and th DIE ORIGINAL DTATENSTRUJTUR BESCHREIBEN For the autencoder using fully connected layers, the input vectors $y_o \in \mathbb{R}$ of size $n_{\text{input}} = n_\xi = 40$ are arranged in the sampling matrix $S_{AE} \in \mathbb{R}^{n_S \times n_\xi}$ as seen in eq. (4.3) resulting in $n_S = 5000$ available samples. Note that the POD uses the same matrix transposed S_{AE}^T as input. In the following hydrodynamic regime will reffer for the input data for knudsen number 10e-4 and rarefied regime will refer to the input data for knudsen numbers 10e2. - insert unshuffled set pls

$$S_{AE} = \begin{bmatrix} f(\xi_1, t_1, x_1) & \cdots & f(\xi_n, t_1, x_1) \\ f(\xi_1, t_1, x_2) & \cdots & f(\xi_n, t_1, x_2) \\ \vdots & \vdots & \vdots \\ f(\xi_1, t_1, x_n) & \cdots & f(\xi_n, t_1, x_n) \\ f(\xi_1, t_2, x_1) & \cdots & f(\xi_n, t_2, x_1) \\ \vdots & \vdots & \vdots \\ f(\xi_1, t_n, x_n) & \cdots & f(\xi_n, t_n, x_n) \end{bmatrix} \quad (4.3) \qquad S_{Conv} = \begin{bmatrix} n_{Filters} & f(\xi_1, \mathbf{t}, \mathbf{x}) \\ n_{Filters} & f(\xi_2, \mathbf{t}, \mathbf{x}) \\ \vdots & \\ n_{Filters} & f(\xi_n, \mathbf{t}, \mathbf{x}) \end{bmatrix} \quad (4.4)$$

Convolutional autoencoders use a different sampling matrix S_{Conv} due to their two dimensional capability resulting in $n_S = 40$ available samples eq. (4.4).$N_{Filters}$ varies over the succeeding layers, growing with the shrinkage of $(\mathbf{t}, \mathbf{x})$. - In the following the data of the BGK full order model with a knudens number of $Kn = 0.00001$, describing a gasflow in the hyrodynamic regime will be reffere to as Π_h. Equally the data of the BGK full order model with a knudsen number of $Kn = 0.01$ will be refferd to as Π_r.

4.1.2 Reduced Basis by POD and Autoencoder

The singular value decomposition of the input X [REF to Section 1] gives the optimal low-rank approximation $\tilde{X}$ of X **??**[Eckard-Young].

$$\underset{\tilde{X}, s.t. rank(\tilde{X})=r}{\text{argmin}} \quad ||X - \tilde{X}||_F = \tilde{U}\tilde{\Sigma}\tilde{V}^* \quad (4.5)$$

Intrinsic variables	3	4	5	6	7	8	9	10
Error	0.0327	0.0153	0.0087	0.0046	0.0021	0.0014	0.0005	0.0003

Table 4.1: L2-Error for different numbers of intrinsic variables for the rarefied gas flow. Calculations with two intrinsic variables are also performed, but not shown here because two intrinsic variables is considered trivial.

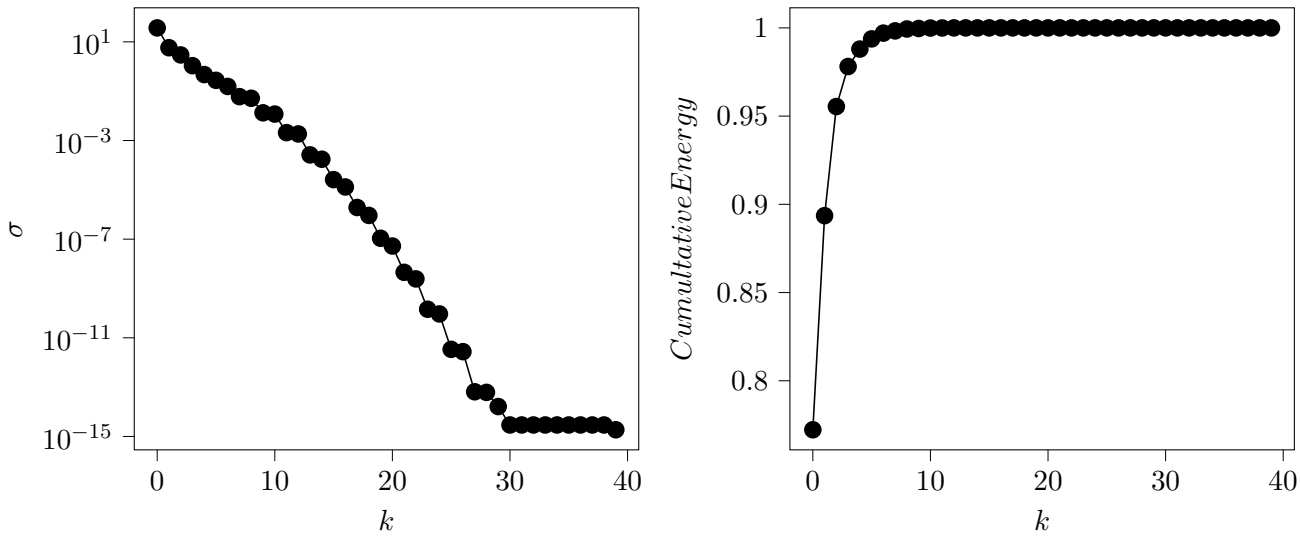


Figure 4.1: Sigular values σ over k number of singular values left and *cumultative energy* over k right for the gas flow in the rarefied regime.

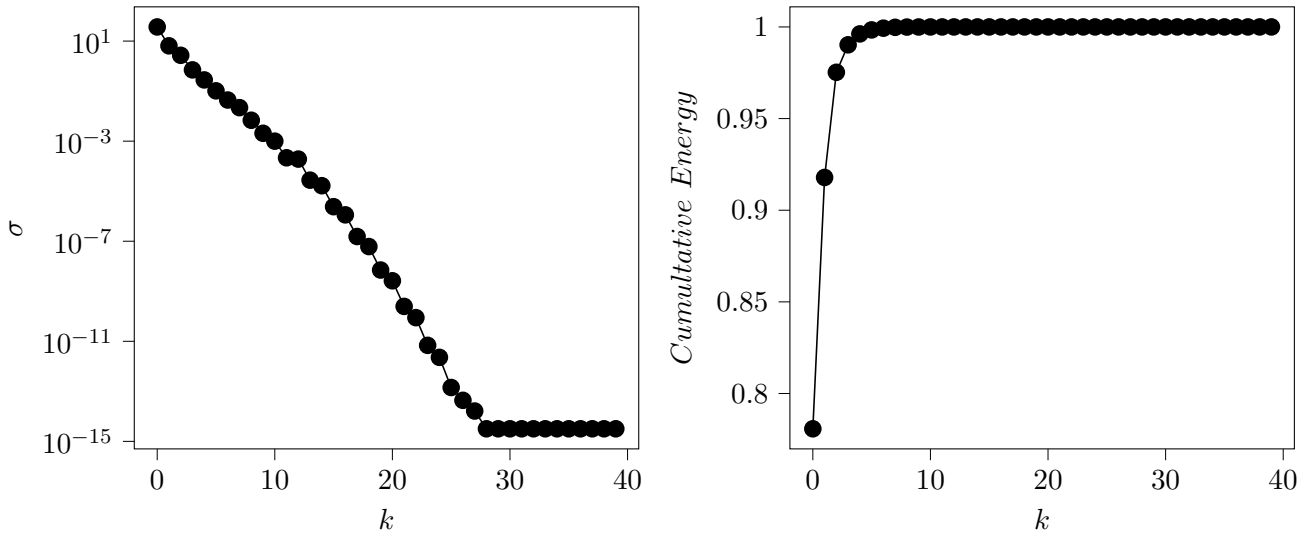


Figure 4.2: Sigular values σ over k number of singular values left and *cumultative energy* over k right for the gas flow in the hydrodynamic regime.

Intrinsic variables	3	4	5	6	7	8	9	10
Error	0.0205	0.0081	0.0030	0.0013	0.0006	0.0002	$6.2e^{-5}$	$2.7e^{-5}$

Table 4.2: L2-Error for different numbers of intrinsic variables for the hydrodynamic gas flow. Calculations with two intrinsic variables are also performed, but not shown here because two intrinsic variables is considered trivial.

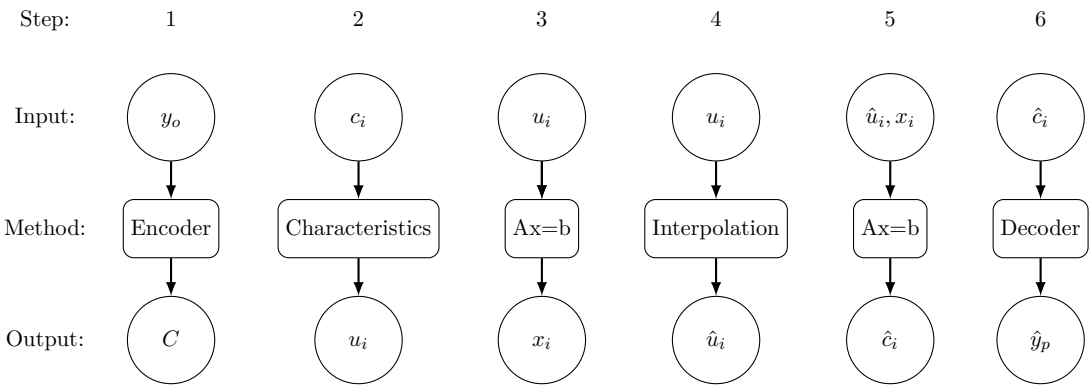


Figure 4.3: This figure shows the steps for obtaining a reduced oder model (ROM). Decoder and Encoder need to be used after training. In step one y_0 is the original input data, C is the Code. In step two c_i is the i-th intrinsic variable and u_i the correspnding characteristic. The eigenvalue problem in step 3 outputs x_i the eigenvector of A, a diagonal matrix composed of u_i and b is the corresponing i-th intrinsic variable c_i. In step 4 $\hat{u}_i$ is the interpolated vector to u_i. Step 5 solves the linear equation for the diagonal matrix A composed of $\hat{u}_i$ times the eigenvector x_i of the eigenvalueproblem in step 3. The output is $\hat{c}_i$ the i-th intrinsic variable corresponding to $\hat{u}_i$ the i-th interpolated characteristic.

4.1.3 Online Phase

loremipsu

4.1.4 Reduced Order Model

The compression of the input data y_0 yields a code $C \in \mathbb{R}^{ix5000}$, composed of the instrinsic variables c_i. The index i corresponds to the i-th intrinsic variable whereas their number is given by the input data. Each of them describes the transport of a discontinuity as seen in **??**. Hence the expolitability of the code in terms of constructing a ROM is not provided. On that account the method of characteristics [16] provides a means to bypass this shortage. It is necessary for $c_i(x,t)$ to satisfy the conservative condition eq. (4.6) and the transport equation eq. (4.7).

$$\frac{d}{dt}\int c_i \; dx = \frac{d}{dt} f_i = const. \qquad (4.6) \qquad\qquad \frac{\partial}{\partial t} c_i + \frac{\partial}{\partial x} f_i = 0 \qquad (4.7)$$

The characteristics u_i describe the constant transport velocities for each variable c_i calculated using eq. (4.8). Subsequently enabling the usage of a simple plynomial interpoaltion of any degree. Furthermore a linear mapping $A_i x_i = c_i$ can be applied for the reconstruction of interpolated code variables $\hat{c}_i$. Figure 4.3 depicts this approach in detail.
Questions concering the capacity of this ROM, e.g. how many samples $\hat{n}_t$ are needed to reconstruct n_t timestamps, are analysed in chapter 5.

$$u_i = \frac{f_i(c_i^-) - f_i(c_i^+)}{c_i^- - c_i^+} \qquad (4.8)$$

5 Results

During the offline phase solutions obtained from the FOM, in this case $\mathbf{H}$ and $\mathbf{R}$, are reduced to their intrinsic dimension, as discussed in chapter 4. From the thereby obtained intrinsic variables which define a reduced basis namely $\mathbf{h}$ and $\mathbf{r}$, the solution can be reconstructed yielding a loss of information. Hence, before buiding a ROM, the reconstruction of $\mathbf{H}$ and $\mathbf{R}$ needs to be evaluated. If the reduction and subsequent reconstruction fails to sustain "important" information, then the reduction algorithm is not suited for building a ROM. What these "important" qualities in the for the BGK model in the test case are and how they can be measured, as well as methods to compare the FOM solution against it's reconstruction is discussed in this section. For the dimensionality reduction the proper orthogonal decomposition (POD), a fully connected neural network (FCNN) and a convolutional neural network (CNN) is used.

A first measure of how much information gets lost is obtained through one of our permanence metrics during neural network training: the MSE over the validation set. Yet, this metric solely applies to neural networks. To be able to compare the POD to it's neural network counterparts another metric is in use: the error over the $L2$-Norm here referred to as thr L_2-Error , already introduced in chapter 3. In table 5.1, the $L2$-Error for all three algorithms on both input data is provided.

Table 5.1: Comparison of the L_2-Error over the reconstructions obtained by POD, FCNN and CNN. The CNN was trained using the k-fold algorithm, therefore the mean μ over $k = 5$ folds, marked with an superscript asterix, is provided. Hence the variance σ^2 for $\mathbf{H}$ is $\sigma^2_{\mathbf{H}} = 2.25 \times 10^{-5}$ and for $\mathbf{R}$ is $\sigma^2_{\mathbf{R}} = 3.61 \times 10^{-5}$. Parenthesised values are obtained from the best fold.

Algorithm	L_2-Error for $\mathbf{H}$	L_2-Error for $\mathbf{R}$
POD	0.0205	0.0087
FCNN	0.0017	0.0019
CNN	0.0142* (0.0095)	0.0178* (0.0097)

The FCNN performs best out of the three both for $\mathbf{H}$ and $\mathbf{R}$ and even drops about one decade compared to POD and CNN for $\mathbf{H}$. Slightly better is the result for $\mathbf{H}$ than for $\mathbf{R}$ using both neural network designs. Yet the performance of FCNN and CNN doesn't seem to change a lot with changing $\mathbf{H}$ and $\mathbf{R}$. In contrast, the POD performs better for $\mathbf{R}$ than for $\mathbf{H}$. The reasons will be discussed hereafter. Note that the CNN is trained with the k-fold algorithm due to a lack of training samples as suggested in [6]. The k-fold algorithm is provided in appendix B. The mean μ over all folds gives an estimate of the models performance. Nonetheless the best performing fold is used in the subsequent analysis, yielding a better performance of the CNN.

Next a step is taken in attempting to estimate the intrinsic dimension $p^\star$ for **R**, which is not set as discussed in chapter 2. To this end the number of intrinsic variables $p^\star$ is varied for POD, FCNN and CNN over $p^\star \in \{1, 2, 4, 8, 16, 32\}$ with both **H** and **R**. Note that the neural networks needed to be trained again for these experiments and that by changing $p^\star$ i.e. widening the bottleneck layer, a gain or loss of capacity occurs which can be connected to stability during training, see chapter 3 and [6]. The stability of the POD, in contrast, is not altered when changing the value of $p^\star$. Shown in fig. 5.1 is the outcome of said experiments. The design for fig. 5.1 is taken from [5].

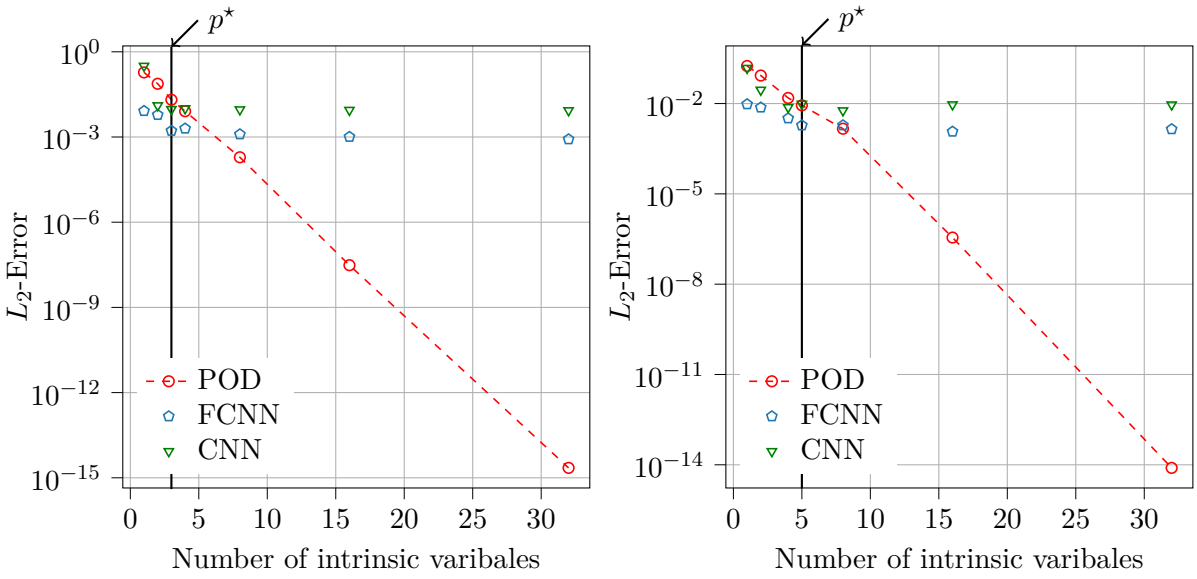


Figure 5.1: Variation of intrinsic variables over the L_2-Error for POD, FCNN and CNN. Results for **H** are displayed on the left and for **R** on the right.

The cumultative energy of the POD modes i.e. $p^\star$ in chapter 4 fig. 4.2 and fig. 4.1 already revealed the performance of the POD with increasing $p^\star$. The loss of information when applying the POD goes linearly to zero with increasing $p^\star$ which is not suprising when consulting the *Eckard-Young Theorem* provided in eq. (4.5) taken from [3]. The left plot of fig. 5.1 displays the results for **H** with $p^\star = 3$ the known dimension of the solution manifold for a simple gas which can be viewed as a continuum, emphasized with a black line. Here a drop of the $L2$-Error can be observed for both neural networks until they meet $p^\star = 3$. Afterwards they stagnate at the $L2$-Error they have reached. Hence for **H** the logically deducted value for $p^\star = 3$ can be confirmed.
The right plot of fig. 5.1 shows the results for **R**.

(The size of both neural networks is kept comparatively small which is the usual case for autoencoders, in order to account for over -and underfitting as explained in [6]. Now any variation of $p^\star$ i.e. the width of the bottleneck layer leads to underfitting when shrinking the capacity as seen for values of $p^\star \in \{1, 2\}$. As stated before the FCNN performs about a decade better than the CNN. Hence when building an autoencoder and working on approaching the optimal capacity for the input data which includes the width of each layer, one obtains an estimator for the input data which is hopefully already at it's optimum. As stated before)

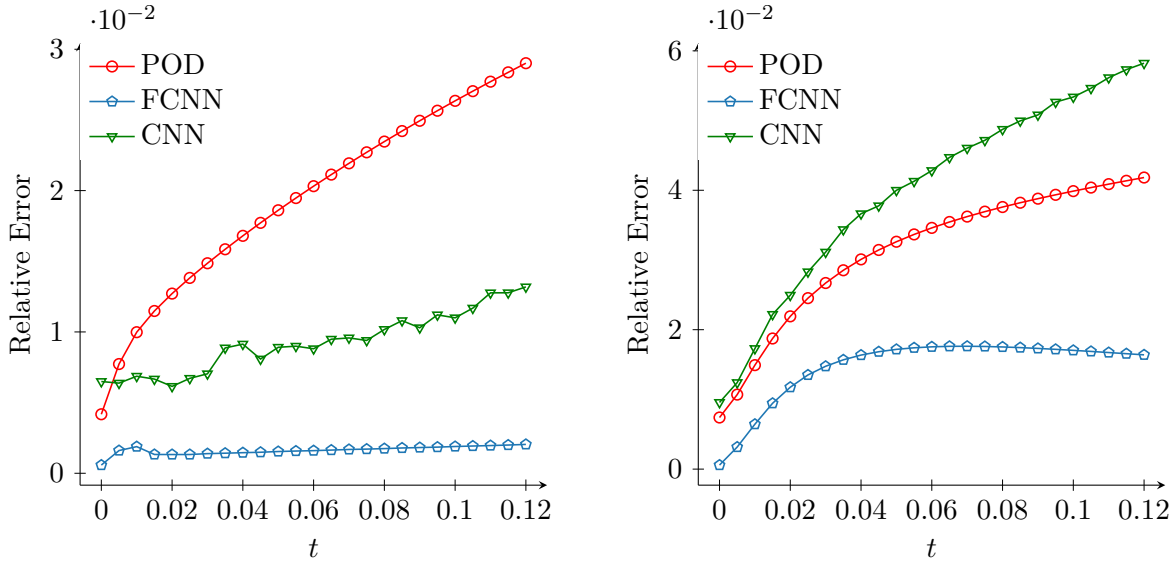


Figure 5.2: Relative Error over time for POD, FCNN and CNN. Results for **H** are displayed on the left, the results for **R** are displayed on the right.

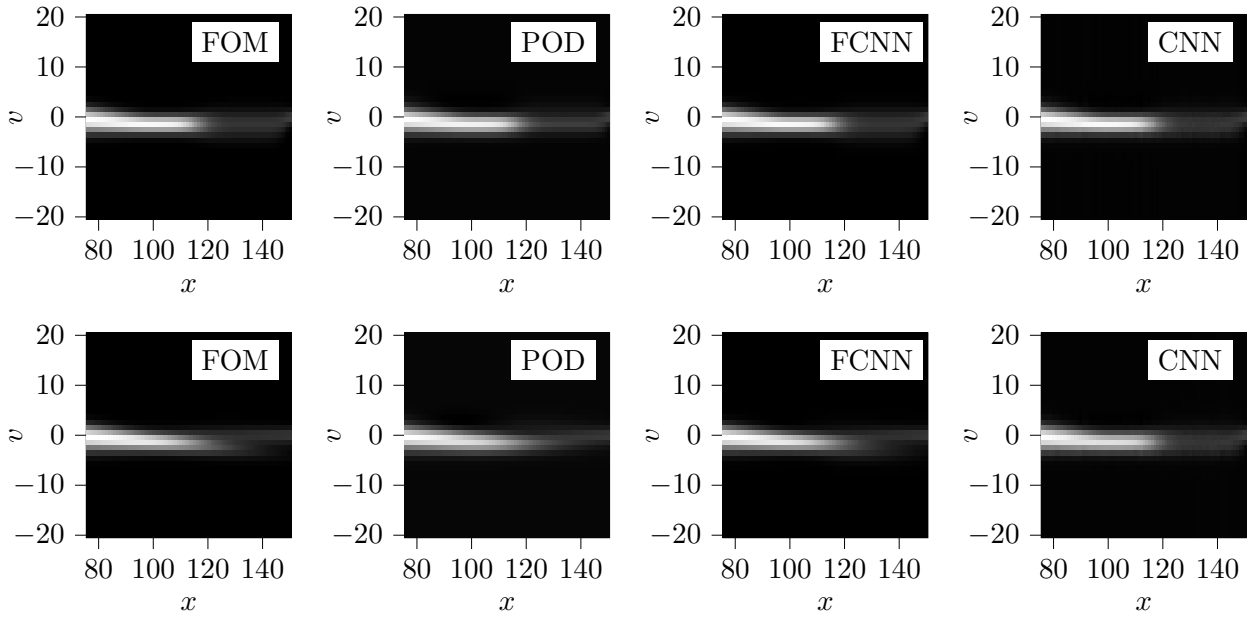


Figure 5.3: Comparison of FOM with three reconstructions obtained from POD, FCNN and CNN. The BGK-model in Sod's shock tube at time $t = 0.12s$ and $x \in [75\text{cm}, 150\text{cm}]$ of the tube, is most difficult to reconstruct by the aforementioned algorithms. Case **H** is displed in the first row, **R** in the second row.

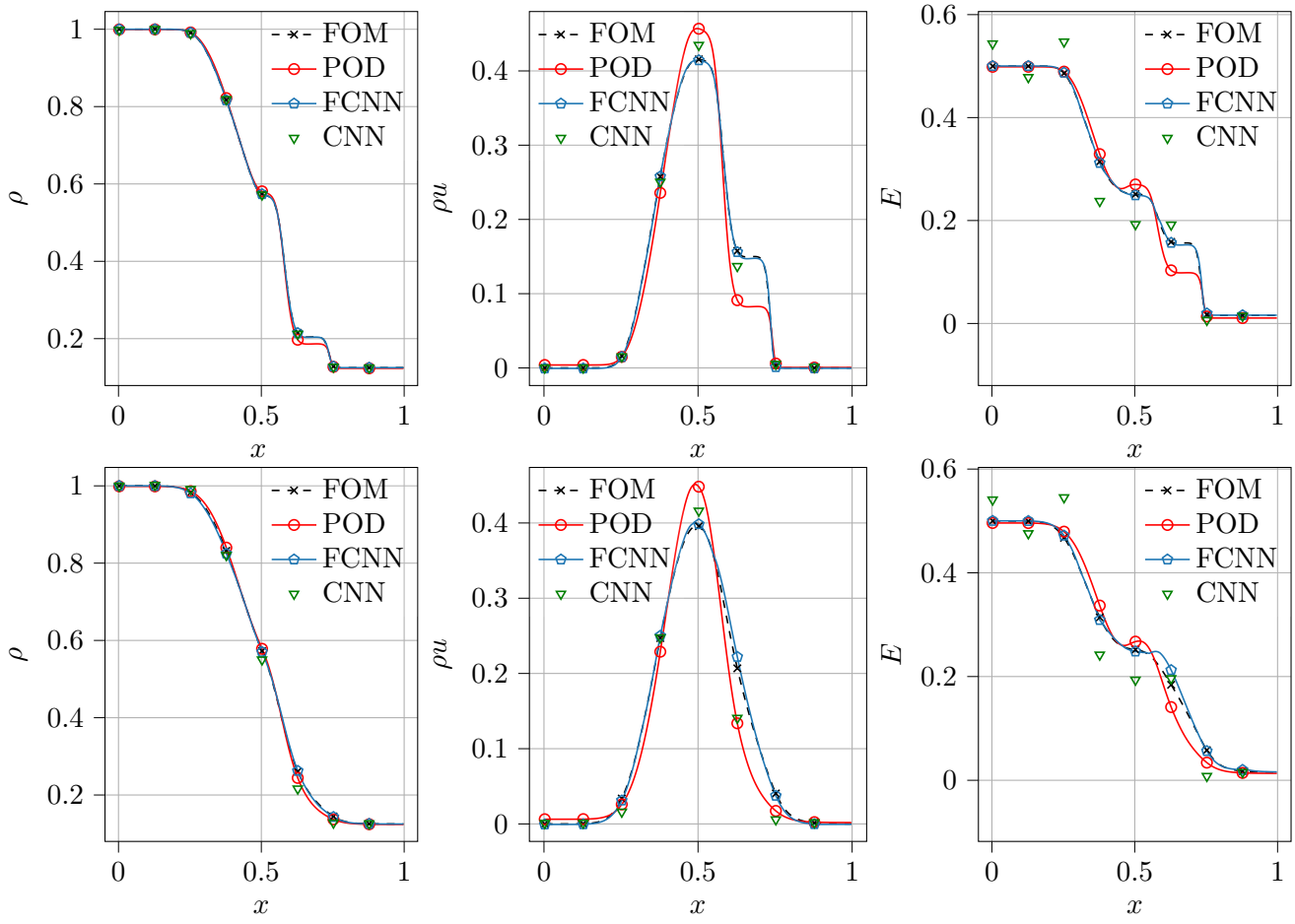


Figure 5.4

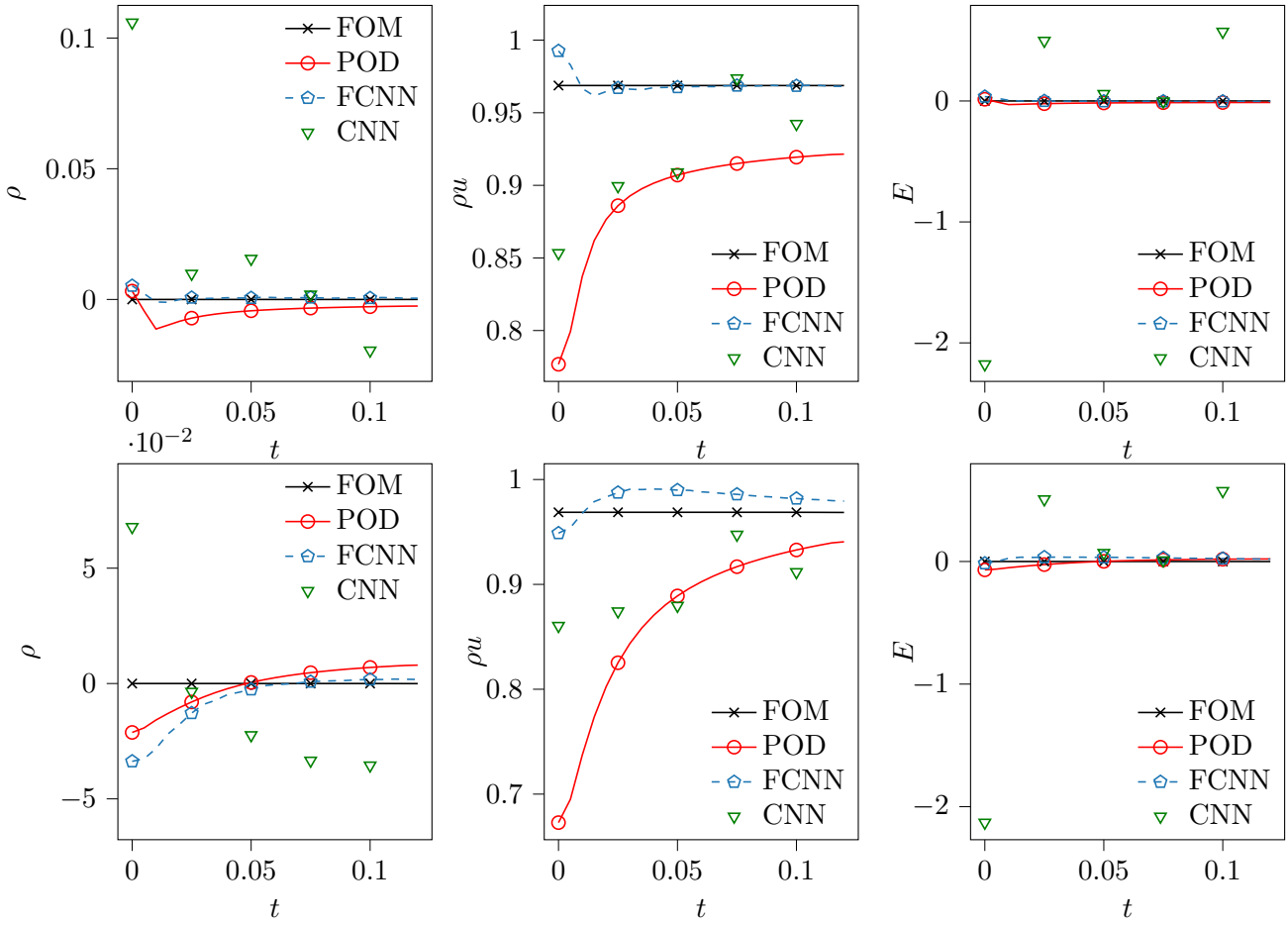


Figure 5.5: Conservation

5.0.1 Hydrodynamic Regime

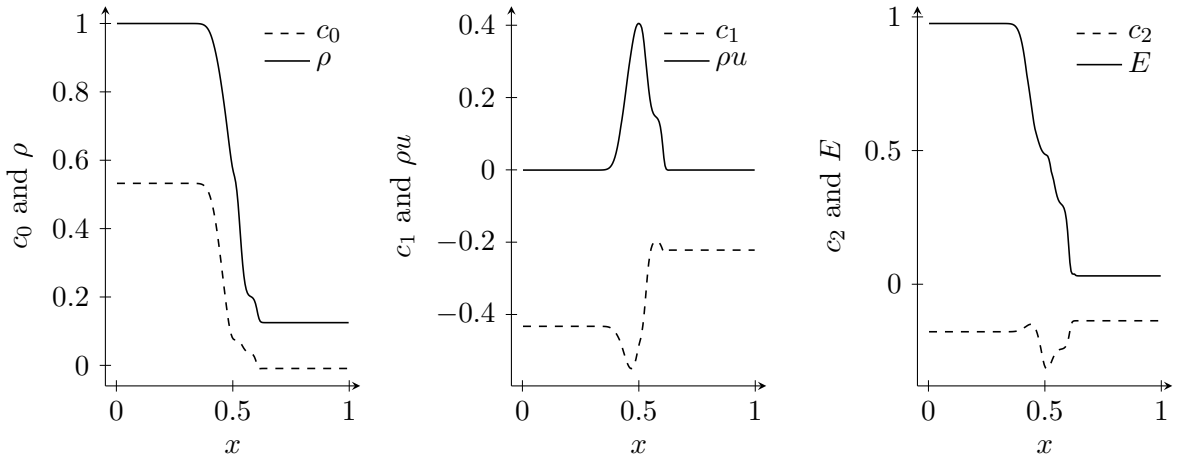


Figure 5.6: Code variables c_1, c_2 and c_3 (dashed lines - -) and macroscopic quantities ρ, E, ρu (full lines –) for $t = 0.05s$.

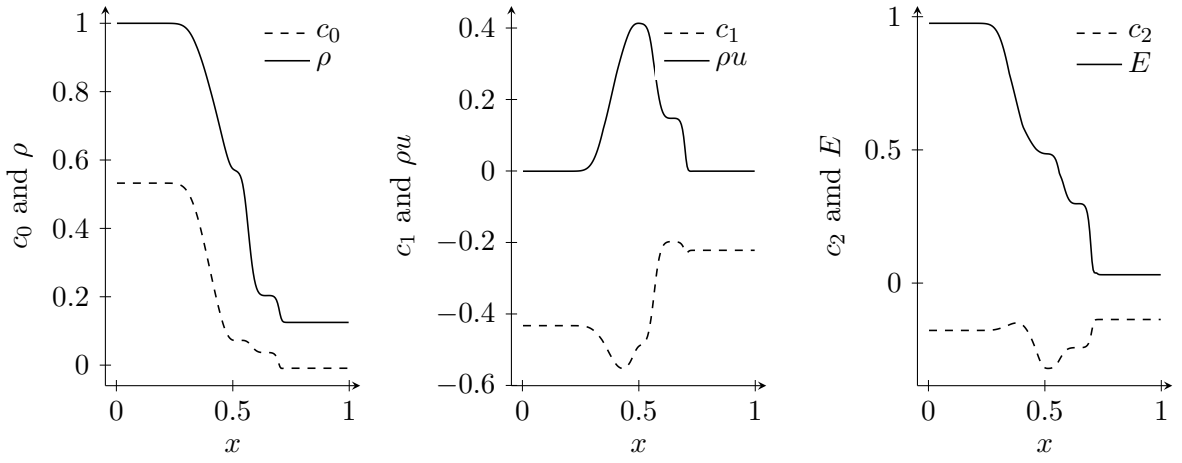


Figure 5.7: Code variables c_1, c_2 and c_3 (dashed lines - -) and macroscopic quantities ρ, E, ρu (full lines –) for $t = 0.099s$.

The number of intrinsic variables defining the rarefied regime is unknown. Therefore experiments varying the number of intrinsic variables from two to ten are performed. Table 5.2 shows the results of two runs. From seven up to ten intrinsic varibales onward the the L2-Error reaches values around $1.2e^{-3}$. From five and six intrinsic variables a shift happens to a value for the L2-Error around $2.3e^{-3}$. A lowering of the number of intrinsic variables up to two increases the L2-Error further. A comparison between results obtained from the SVD and the autoencoder, show that both algorithms perform nearly the same with eight intrinsic variables with an L2-Error of $1.4e^{-3}$. With more than eight intrinsic variables

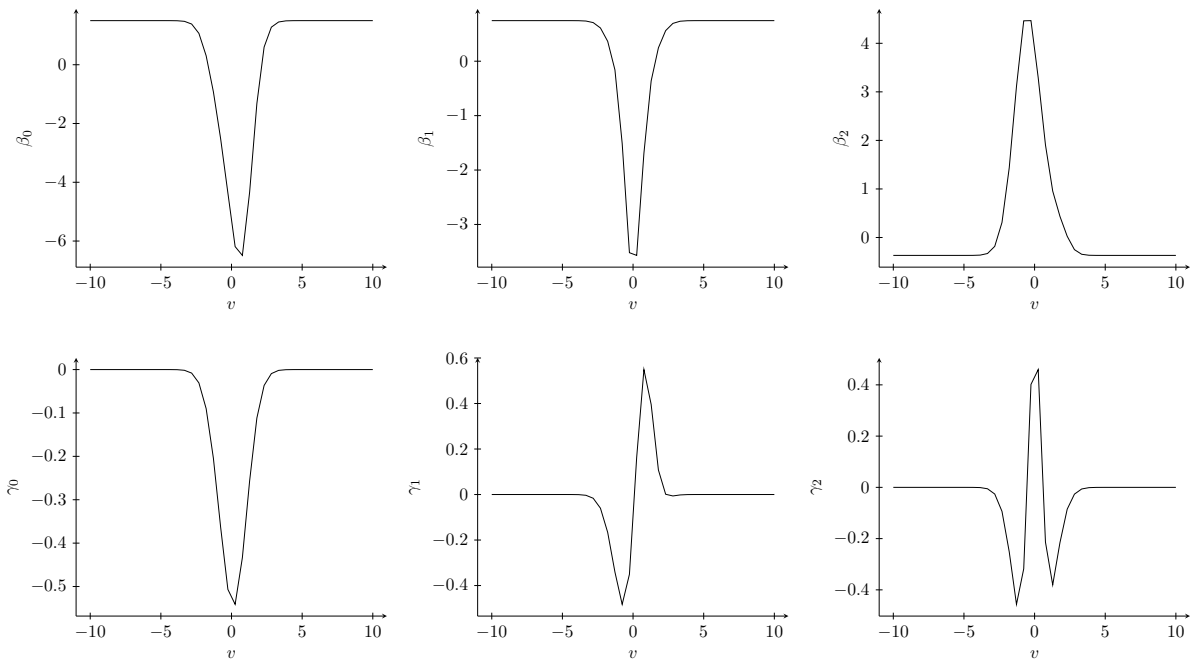


Figure 5.8: Comparison of the intrinsic variables generated by POD γ_1, γ_2 and γ_3 with the intrinsic variables of the convolutional autoencoder β_1, β_2 and β_3.

Intrinsic variables	3	4	5	6	7	8	9	10
Error 1st run	0.0048	0.0025	0.0026	0.0027	0.0013	0.0014	0.0013	0.0009
Error 2nd run	0.0038	0.0024	0.0016	0.0015	0.0011	0.0010	0.0012	0.0010

Table 5.2: L2-Error for different numbers of intrinsic variables for the rarefied gas flow. Experiments with two intrinsic variables are also performed, but not shown here because two intrinsic variables is considered trivial.

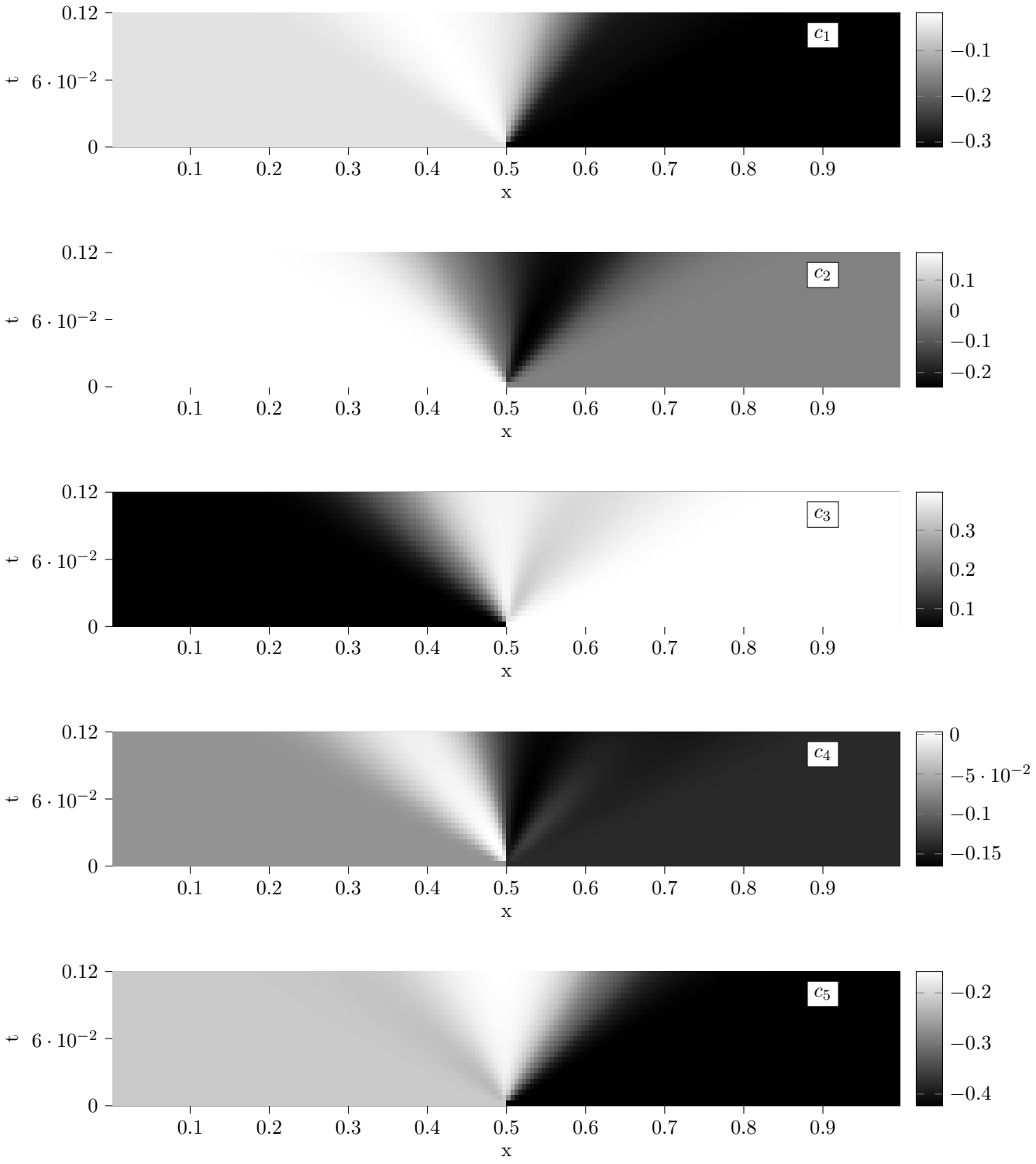


Figure 5.9: Intrinsic Variables of **R**

5.0.2 Discussion and Outlook

Appendix

A Hyperparameters for the Fully Connected Autoencoder

To start with a working model a guess is needed to be made about the initial design of the architecture. Here the hyperparameters already found are used as a starting point and shown in Table A.1.

To start with the number of layers or depth as described in **??** is determined, as they set a

Mini batch size	Intrinsic dimensions	Epochs	Learing rate	activation code/rest
16	3	2000	1e-4	Tanh/LeakyreLU

Table A.1: Initial hyperparameter selection

consequential part of the representational capacity of the model and therefore can initiate over- and underfitting at an early stage of the parameter search. **??** and **??** in **??** show the training and validation error for five designs shown in table A.2.

For the hydrodynamic regime a number of layers greater than four results in a slight overfitting

Number of layers	Reduction per layer
10	$40 \to 40 \to 20 \to 10 \to 5 \to 3$
8	$40 \to 40 \to 20 \to 10 \to 3$
6	$40 \to 40 \to 20 \to 3$
4	$40 \to 40 \to 3$
2	$40 \to 3$

Table A.2: Initial hyperparameter selection

at an early stage of training (at around 100 epochs). Below four layers an underfitting can be observed. Hence yielding the conclusion, that four layers result in the best perfoming net at this early stage. Overfitting occurs only after the 1000th epoch and is less than with the other three nets that show overfitting. Note, that contrary to expected results, the train error is lower than the test error. The random shuffling during preprocessing might be taken into account here. Nonetheless solely overfitting is defined by differences between train and test loss. In Addition four layers and lower show a stable training in relation to the rest.

The analysis of the number of layers for the rarefied regime is not showing overfitting as obvious as before. The network with two layers underfits in contrast to the other nets whereas nets with more than two layers reach a similar train - and test loss. Networks with more than 4 layers, again show an unstable training compared to the net with four layers. Train and test loss show a diverging behaviour after around the 100th epoch.

In conclusion the net with four layers performs best for both the hydrodynamic and the rarefied regime.

Training duration is raised form 2000 epochs to 5000, as training did now converge completely as seen in **??** and fig. A.2.
In the following the width of the hidden layer will be analysed. There are two available hidden layers for the autoencoder. But as the architecture of the decoder mirrors that of the encoder only one parameter needs to be varied. For both the hydrodynamic and the rarefied regime five experiments are conducted, varying the hidden units from ten to fifty. Results for Π_h and Π_r are shown in table A.3. For **H** and **R** forty hidden units performs best with an error around $1e^{-3}$ and

Hidden units	10	20	30	40	50
Error for Π_h	0.0054	0.0027	0.0036	0.0017	0.0032
Shrinkage factor for **H**	0.3	0.1	0.015	0.075	0.06
Error for Π_r	0.0078	0.0027	0.0022	0.0015	0.0025
Shrinkage factor for **R**	0.5	0.25	$0.01\overline{6}$	0.0125	0.01

Table A.3: L2-Error for different number of hidden units for Π_h and Π_r.

a shrinkage factor of 0.075 for **H** and 0.0125 for **R**. The combination of representational capacity set by the number of hidden units and shrinkage factor has an optimum here. When decreasing the number of hidden units, the representational capacity of the model also decreases. At the same time the shrinkage factor increases, therefore less information needs to be compressed in one step. The error increases with deceasing the number of hidden units from forty. This can be explained by the lower representational capacity. When increasing the number of hidden units to fifty the error also increases though, the representational capacity grows. Here the decreased shrinkage factor could be responsible for that.
Next the mini-batch size is analysed. Results are displayed in table A.4 for **H** and in table A.5 for **R**. Additionally appendix B.0.1 shows the training for both in put data **H** and **R**. For both input data experiments are first conducted with mini-batch sizes in the range of : $[2, 4, 8, 16, 32]$.
The results of the L2-Error show best performance between a mini-batch size of 8 and 16 for **H**. Likewise the results of the L2-Error for **R** show best perfomance between mini-batch sizes of 4 and 16. Therefore additional experiments are conducted. Three additional experiments for **H** with mini-batch sizes of: $[10, 12, 24]$. Two additional experiments for **R** with mini-batch sizes of: $[6, 10]$. The training reveals that smaller batch sizes lead to oscillating traing and test errors. This can be overcome by a smaller learning rate, but also leads to increased computational time as more epochs are needed to achieve a comparable training and test loss at the last epoch. A mini-batch size of 8 is bordering the zone where the oscillations of training and test loss subside. At the same time a mini-batch size of 8 to 10 indicate also a growth of the L2-Error. In conclusion a mini-batch size of 8 is chosen to continue to working with for **H** and **R**. The osccilations which make the training instable can be battled with a lower learning rate as soon as training starts to tremble.
Together with the mini-batch sizes, activations are examined. Hence experiments with different

Batch Size	2	4	8	10	12	14	16	32
Error for Π_h	0.0011	0.0011	0.0011	0.0029	0.0018	0.0018	0.0013	0.0030

Table A.4: L2-Error for different mini-batch sizes for Π_h.

activation functions are performed with the innitially used mini-batch size of 16. Besides the mini-

Batch Size	2	4	6	8	10	16	32
Error for Π_r	0.0014	0.0010	0.0012	0.0012	0.0014	0.0017	0.0017

Table A.5: L2-Error for different mini-batch sizes for Π_r.

batch size, epochs are as well taken from the inital selection of hyperparameters as 2000 epochs. Unlike the previous two examinations the selection of activation functions shows early in training if a acceptable level of perfomance can be achieved. This is a personal observation made throughout working on this thesis. First five activations are applied to all the four layers. Second a combination of activation functions is studied, where the intrinsic variables is activated with a different function than the remaining layers. Results can be observed in table A.6 and table A.7. Both input data

Activation function	ReLU	ELU	Tanh	SiLU	LeakyReLU
Error	0.0028	0.0019	0.0036	0.002	0.0039
Activation function	ELU/Tanh	LeakyReLU/Tanh	ELU/SiLU		
Error	0.0019	0.0017	0.0019		

Table A.6: L2-Error different activation functions and combinations for **H**.

Activation function	ReLU	ELU	Tanh	SiLU	LeakyReLU
Error	0.0048	0.0029	0.0037	0.0134	0.0.0034
Activation function	ELU/Tanh	LeakyReLU/Tanh	ELU/SiLU		
Error	0.0.0032	0.0030	0.0036		

Table A.7: L2-Error different activation functions and combinations for **R**.

lead to different results this time. For that reason **H** and **R** are studied separately. Starting with **H** and the values of the L2-Error, it can be observed that, while ELU and SiLU stand out with the lowest value of the L2-Error, when applying the same activation function to all layers, all three coupled activations convince in the combination case. Applying different activations to the layers increases the representational capacity of the model since the model can feed on a greater variety of functions as explained in the previous subsection. The need for a great representational capacity reduces for **H**. The reasons are the linearity of that case as described in [1]. Hence two activations (ELU and SiLU) from a similar family of functions perform well on **H** without the need of a second actiation. Still adding another activation render as good results concerning the L2-Error. When observing the train -and test loss in fig. A.5 one can notice that overfitting clusters the activations into two groups. While they all achieve a similar MSE-Loss for train -and test data only ELU and Tanh are not sowing overfitting. The combination of two activations functions yields an abundance of overfitting only for the combination of LeakyReLU with Tanh. In addition LeakyReLU with Tanh reaches together with ELU in combination with SiLU the lowest train -and test loss. The slight overfitting observed with the combination ELU and SiLU give rise to choose for the final model LeakyReLU with Tanh.
To continue with **R** the L2-Error is again observed in table A.7, ELU stands out for the single activation. ELU with Tanh and LeakyReLU with Tanh meet the lowest L2-error for the combination of activations. Next observing the training for the three activations in fig. A.6 one finds that all

three show little to no overfitting, while the combination of LeakyReLU and Tanh deviates from that slightly. From this point no clean decision can be made which activation or combination to take. Therefore another run is conducted for the three. Results are shown in table A.8. Only LeakyReLU with Tanh and ELU alone convince with a lower L2-loss. The train -and test loss does not give any hint which of the two remaining activations/ combination of activations to take concerning overfitting or a profound gradient for the train -and test loss. Hence a third run is performed taking the parameters already produced for the two models and training them for another 2000 epochs. This warm-starting of the models finally gives an answer on which one to take. The results in table A.8 show that LeakyReLU with Tanh, just as for **H**, perform well on **R** with the given model. In fig. A.7 the train and test loss during training are provided for teh second and third run.

Activation function	ELU	ELU/Tanh	LeakyReLU/TanH
Error 2nd run	0.0034	0.0029	0.0025
Error 3rd run		0.0024	0.0019

Table A.8: L2-Error different activation functions and combinations for **H**.

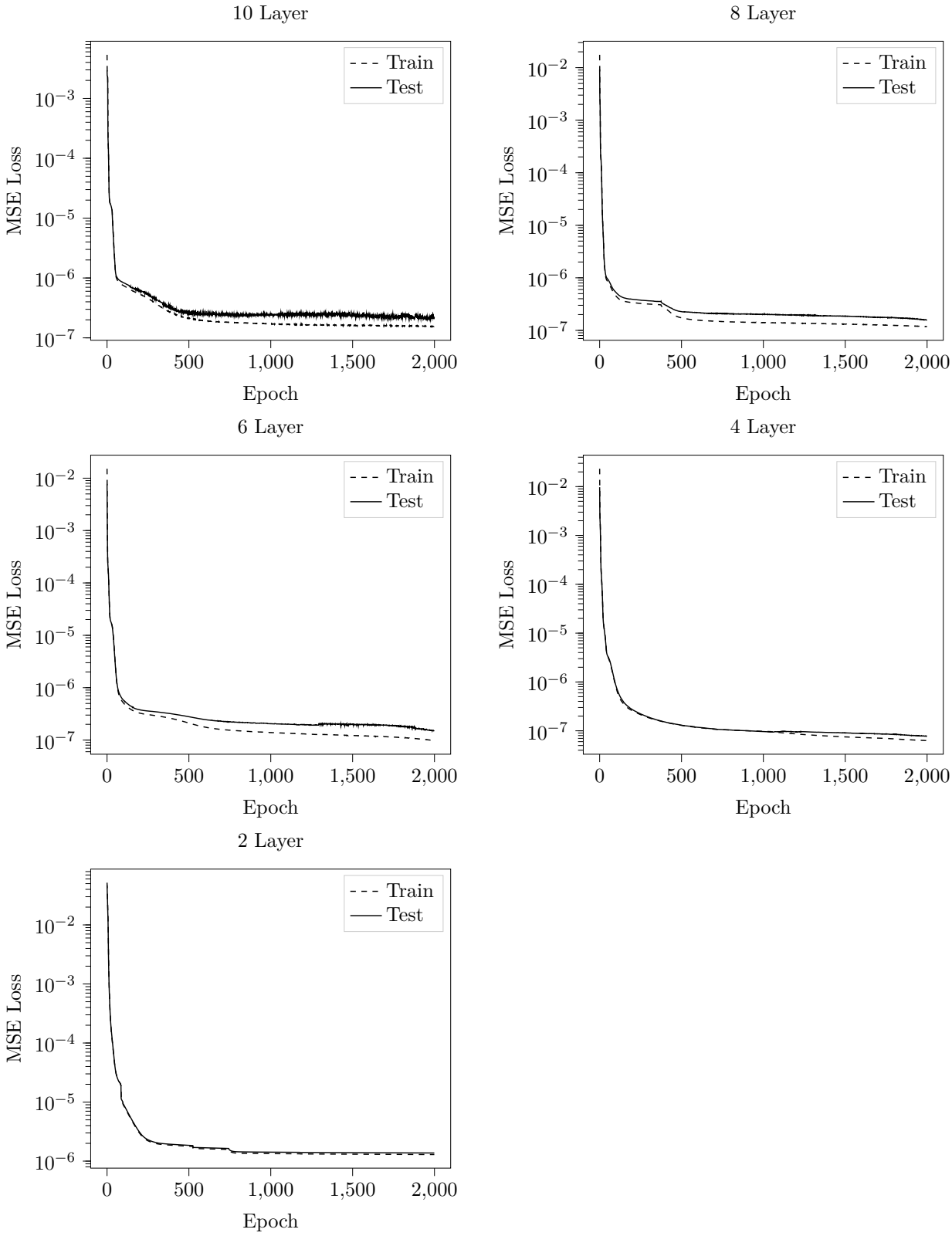


Figure A.1: Experiments with five different depth on **H**. Training and test loss are shown over 2000 epochs.

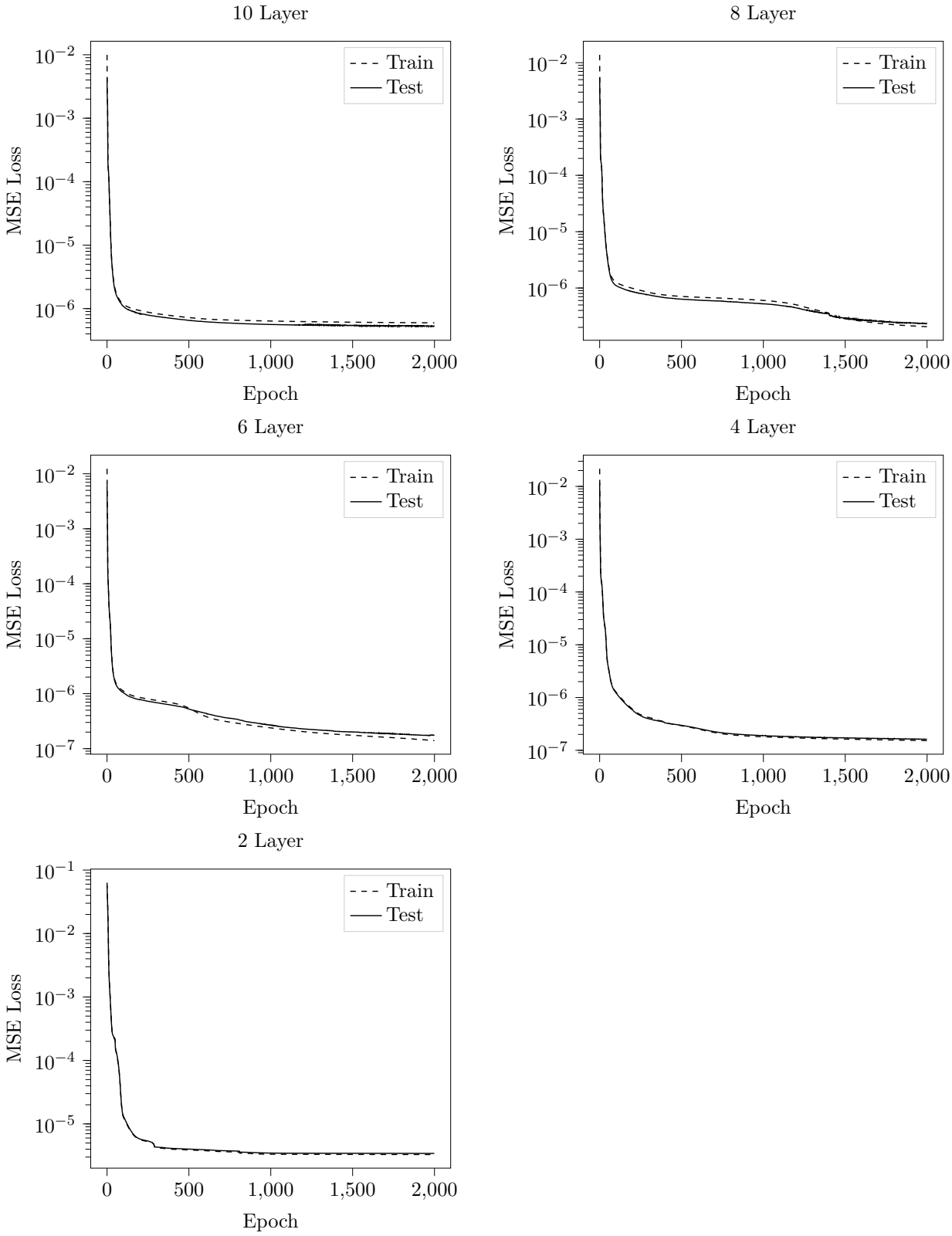


Figure A.2: Experiments with five different depth on **R**. Training and test loss are shown over 2000 epochs.

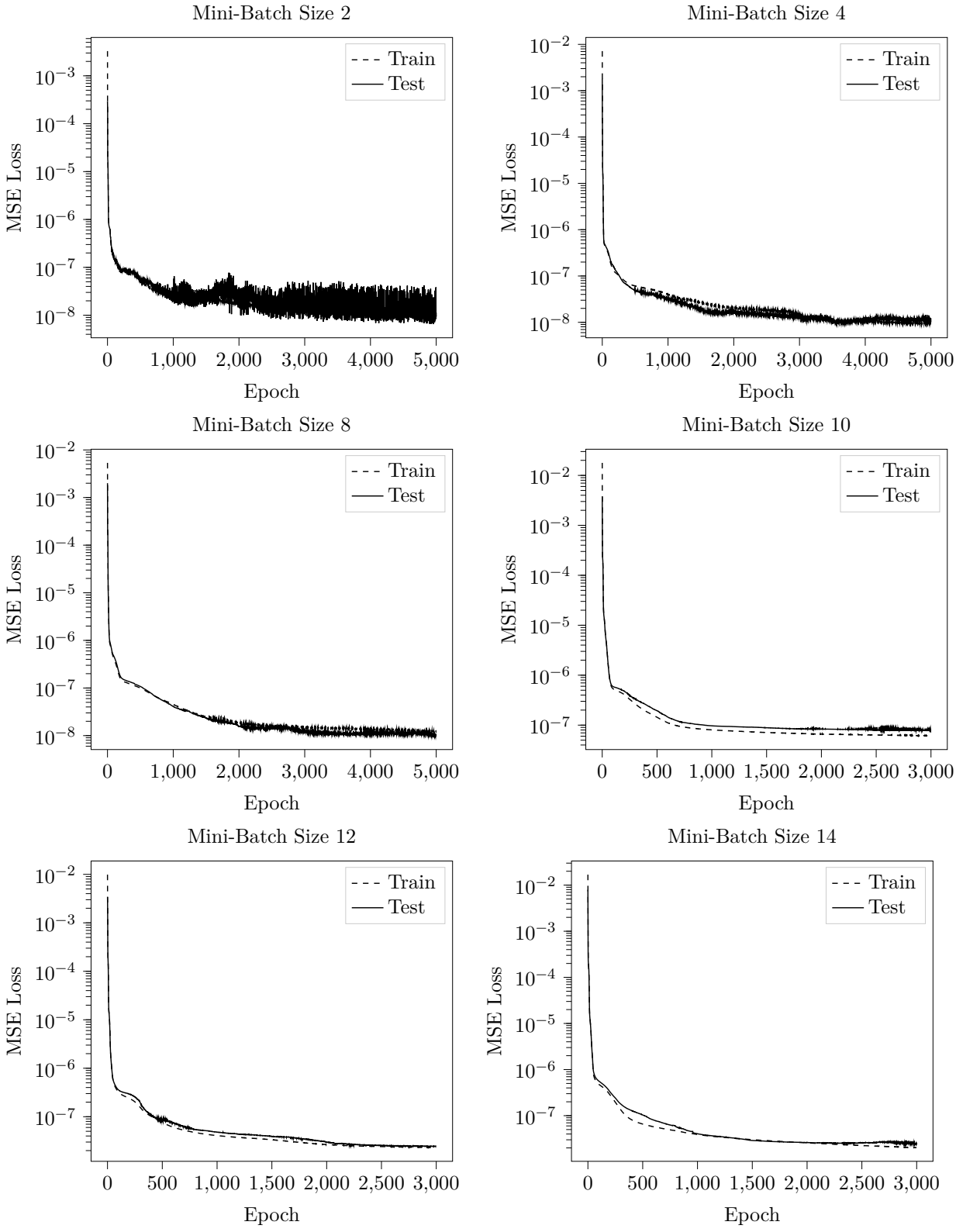


Figure A.3: Training with different mini-batch sizes for **H**. Train -and test loss is shown over 5000 epochs.

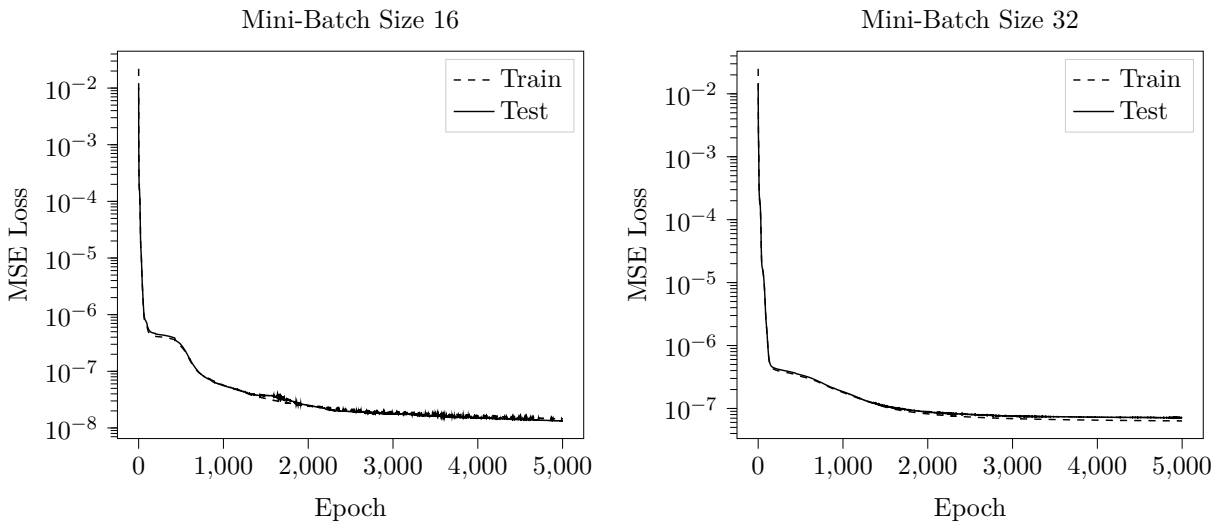


Figure A.3: Training with different mini-batch sizes for **H**. Train -and test loss is shown over 5000 epochs.

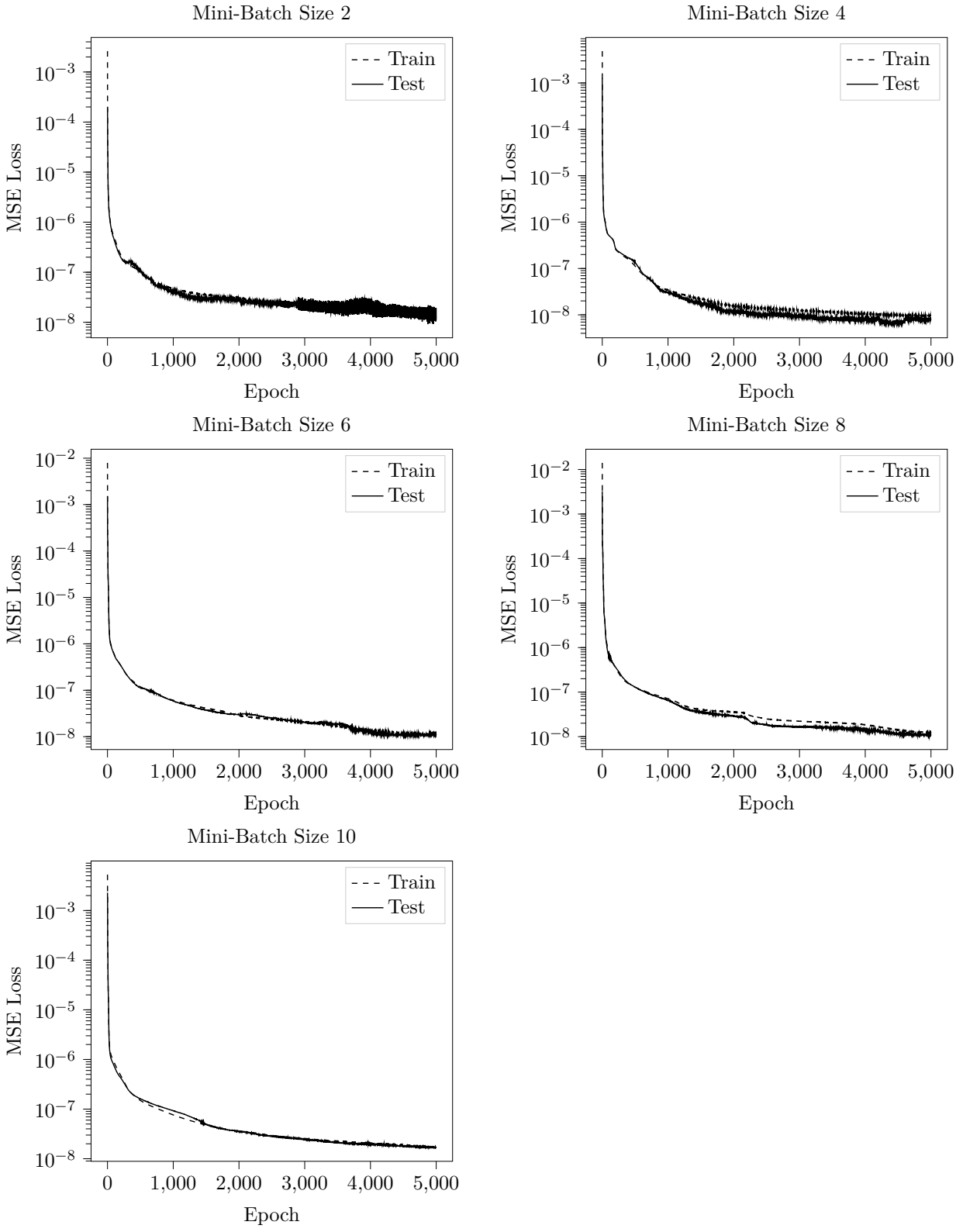


Figure A.4: Training with different mini-batch sizes for **R**. Train -and test loss is shown over 5000 epochs.

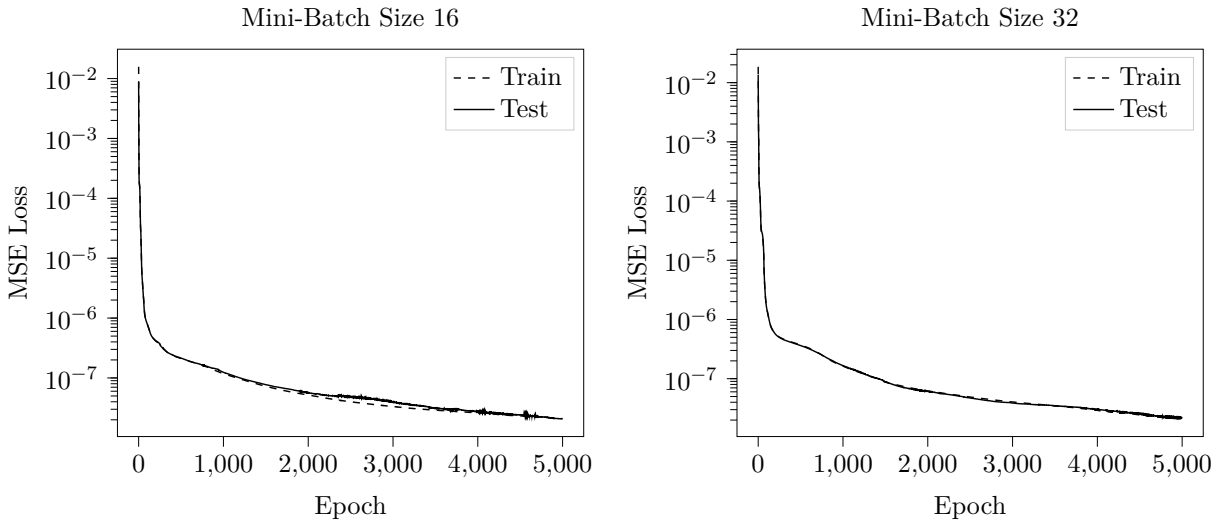


Figure A.4: Training with different mini-batch sizes for **R**. Train -and test loss is shown over 5000 epochs.

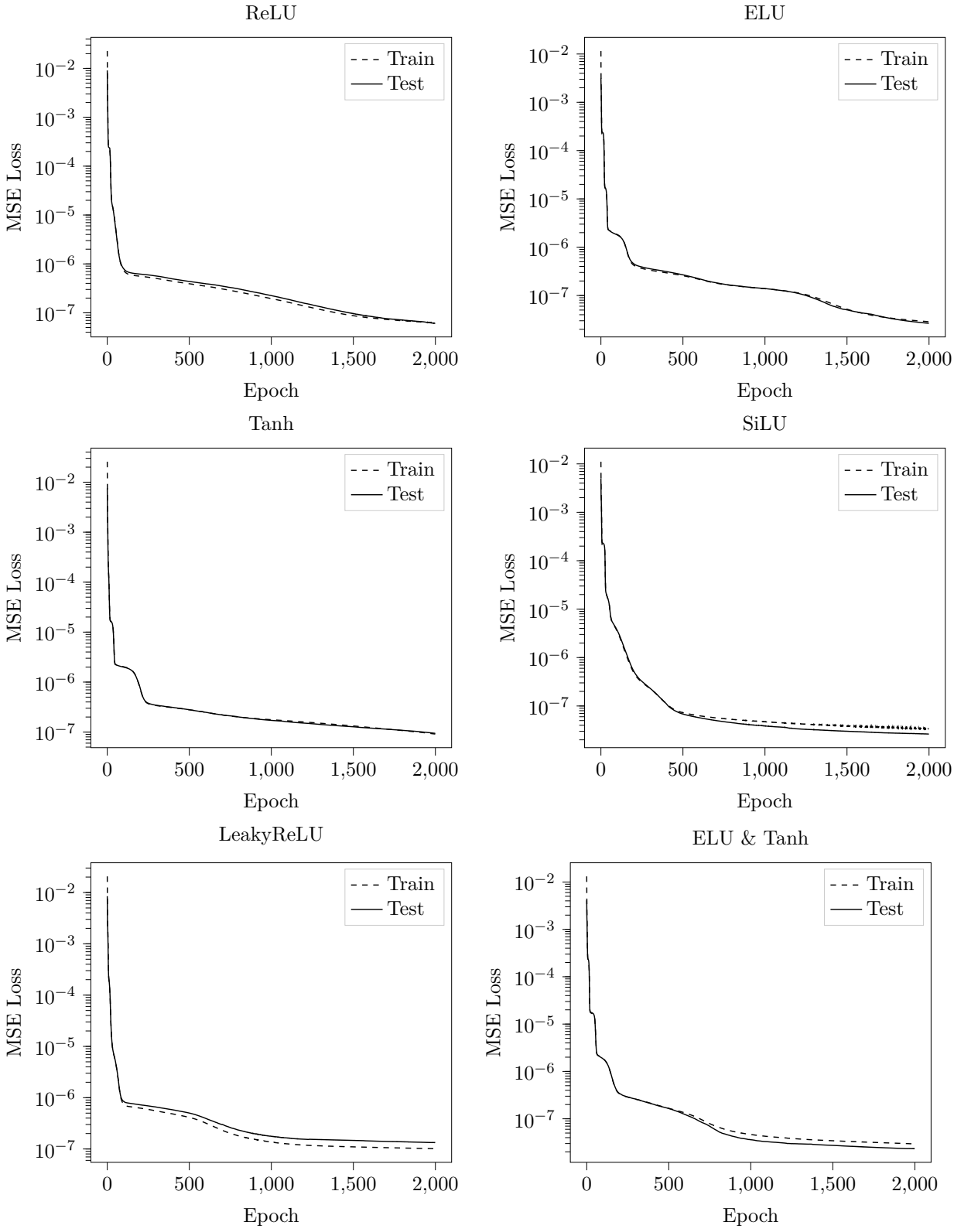


Figure A.5: balblabla

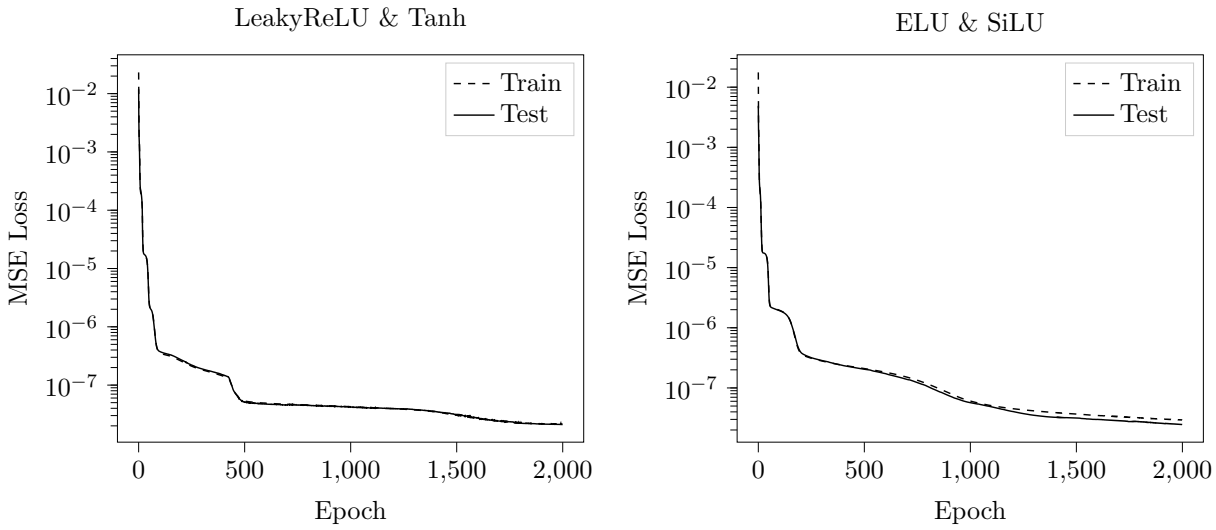

LeakyReLU & Tanh
ELU & SiLU
Train
Test
MSE Loss
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0
500
1,000
1,500
2,000
Epoch

Figure A.5

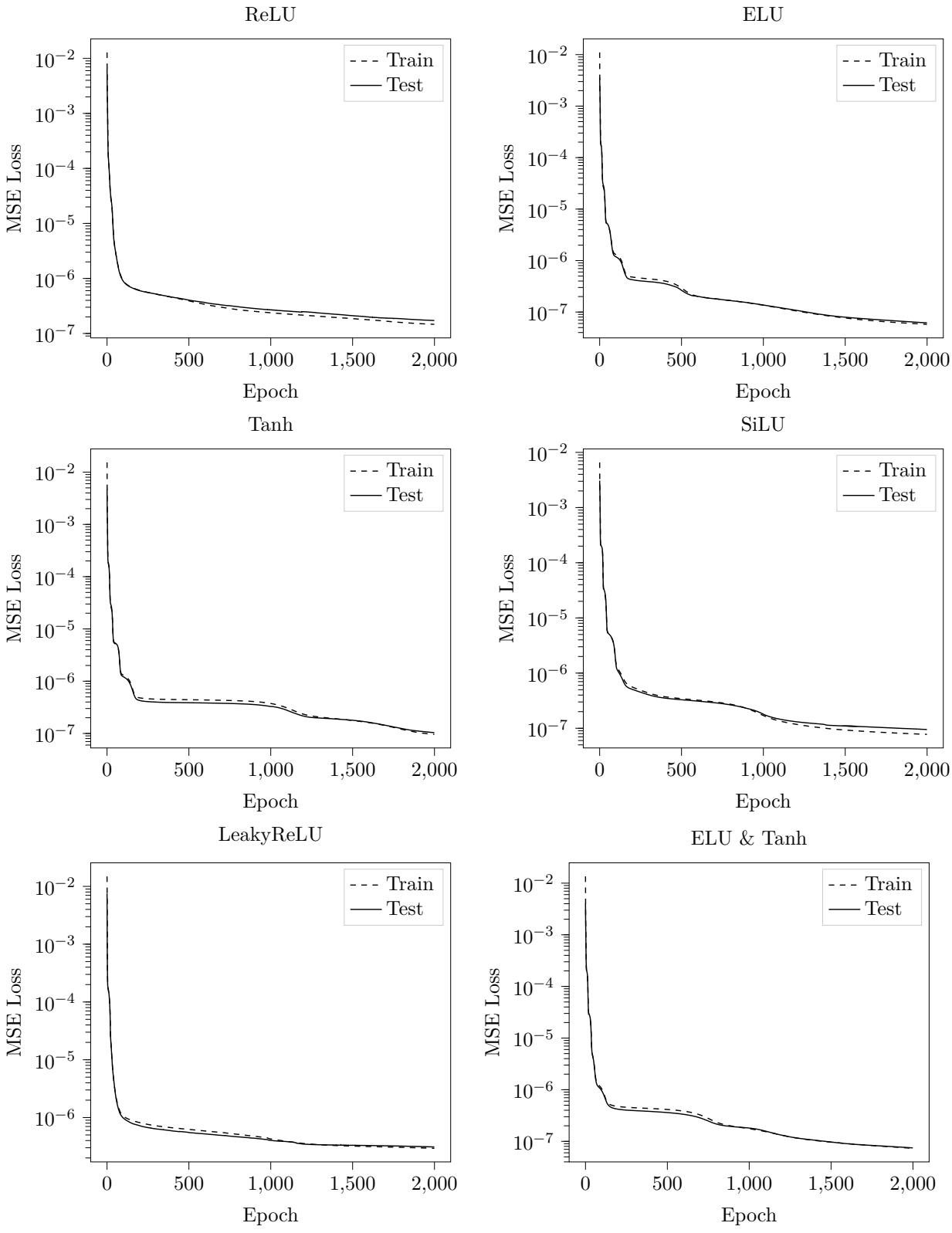


Figure A.6: blablabla

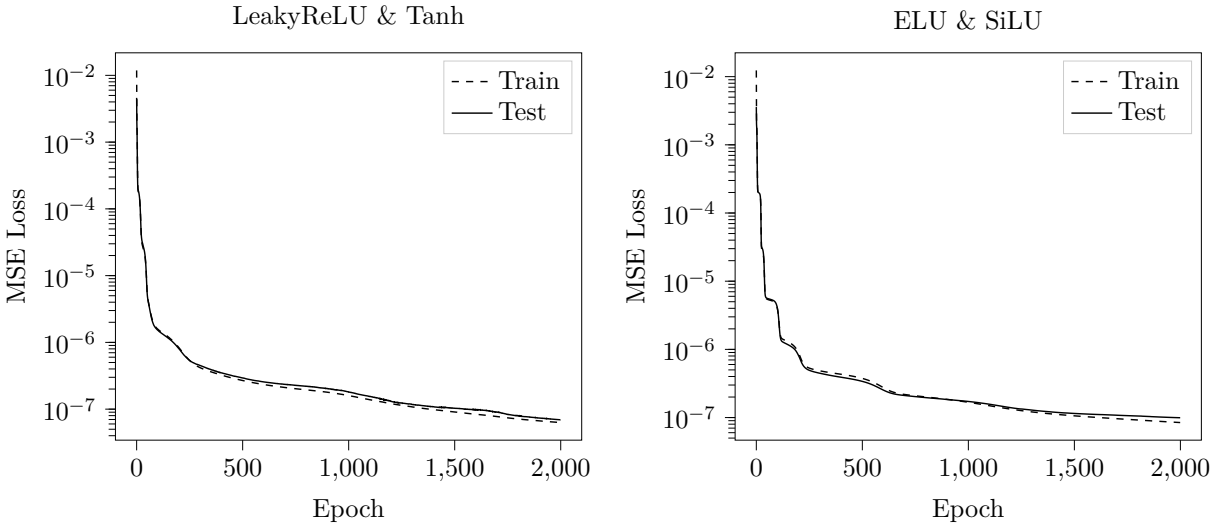


Figure A.6: blablabla

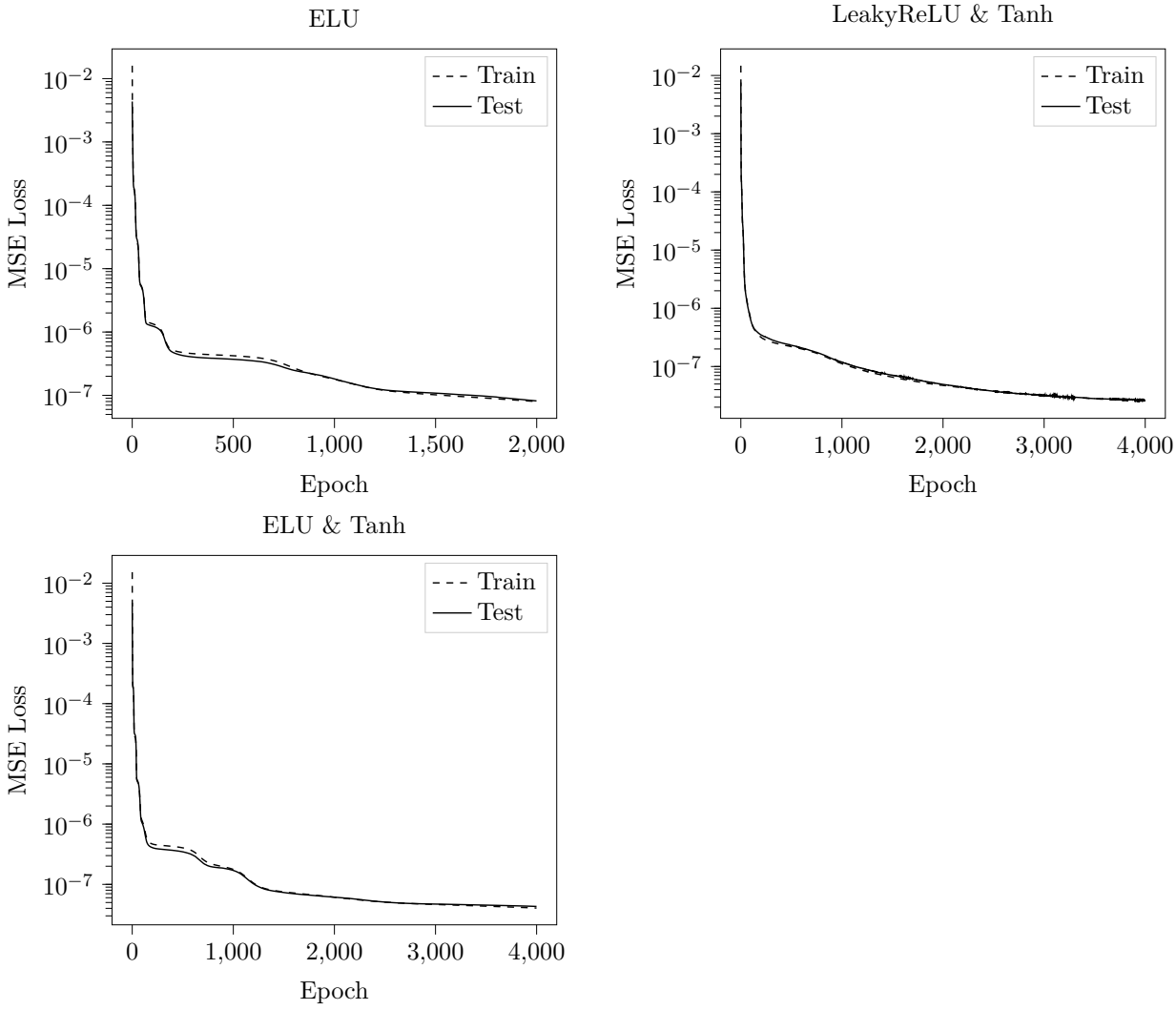
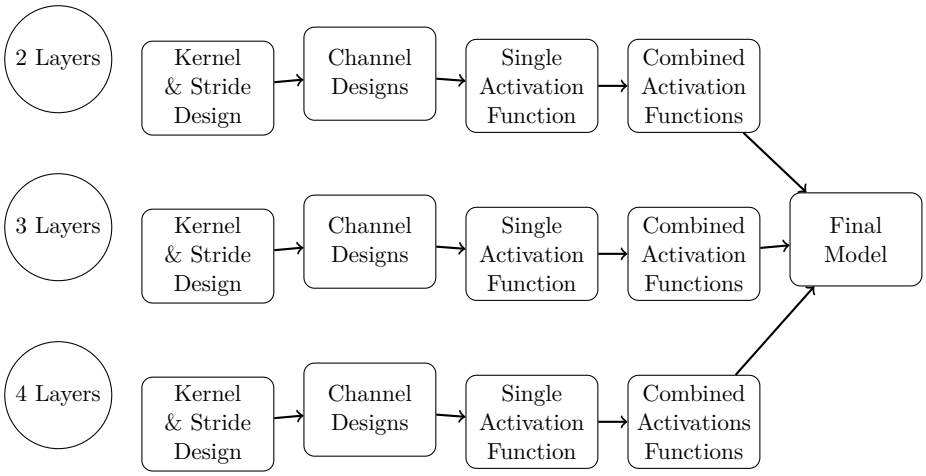


Figure A.7: blablabla

B Hyperparameters for the Convolutional Autoencoder



B.0.1 Appendix B

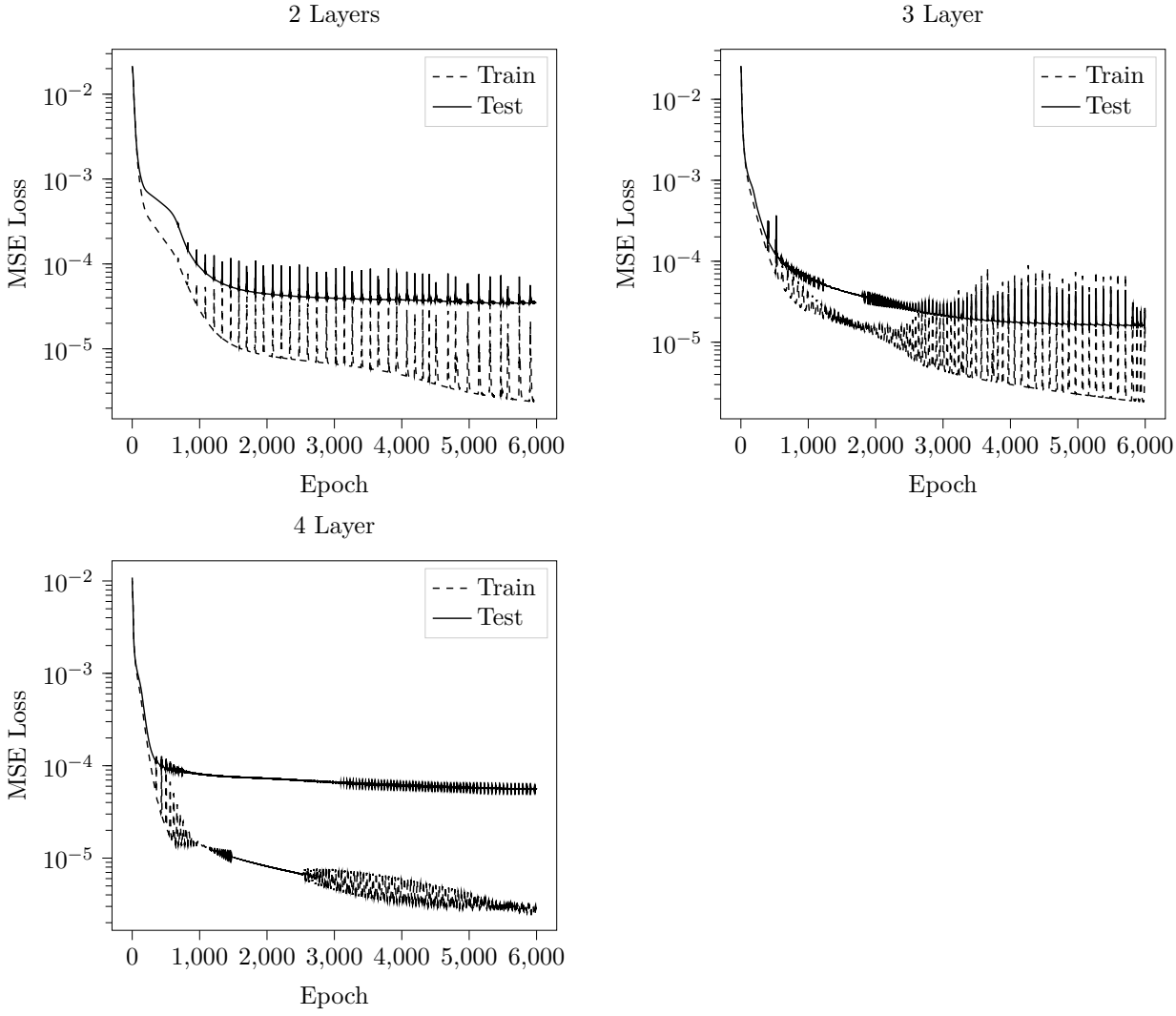


Figure B.1: Three convolutional networks with differing depth and with identical kernel evolution for **R**.

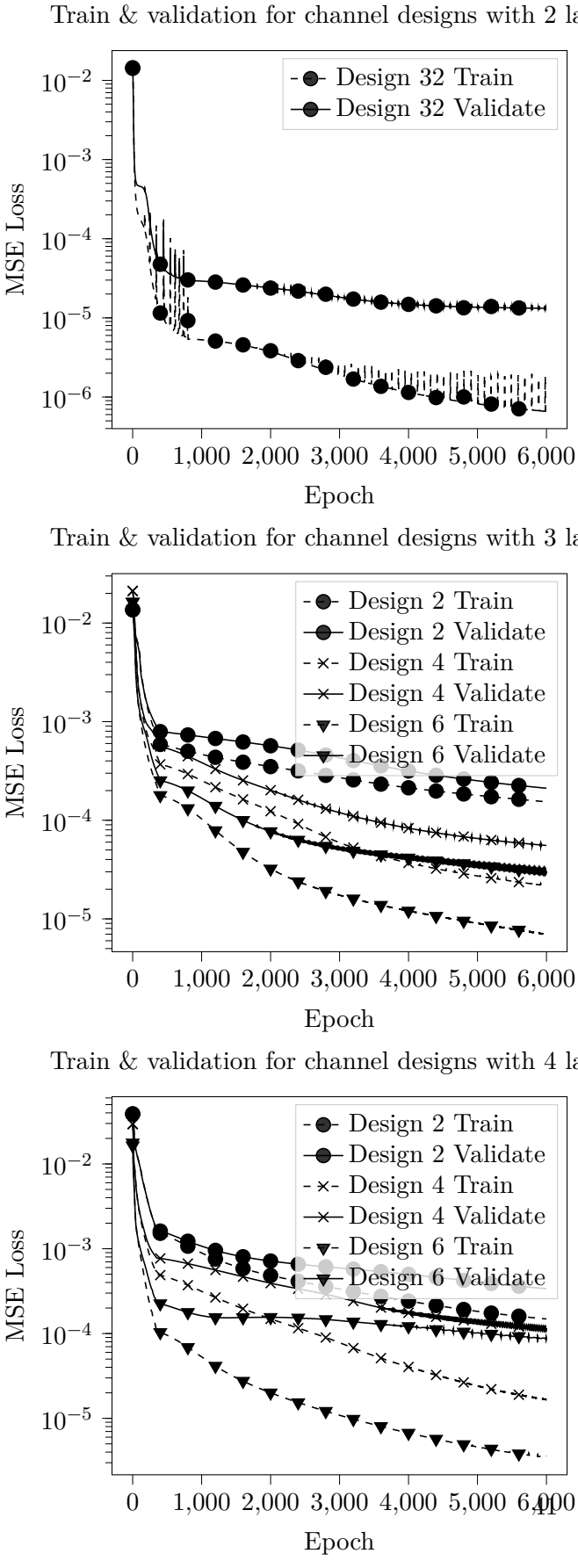


Figure B.2: Different Channel sizes for three convolutional networks with differing depth for **R**.

Bibliography

[1] Bhatnagar, Gross and Krook, *A model for collision processes in gases*, .

[2] T. Franz, *Reduced-order modeling of steady transonic flows via manifold learning*. 2016.

[3] S. L. Brunton and J. N. Kutz, *Data driven science and engineering*. 2019.

[4] F. Bernard, A. Iollo and S. Riffaud, *Reduced-order model for the bgk equation based on pod and optimal transport*, .

[5] K. Lee and K. T. Carlberg, *Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders*, .

[6] I. J. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[7] D. Rumelhart, G. Hinton and R. Williams, *Learning internal representations by error propagation*, .

[8] D. H. Ballard, *Modular learning in neural networks*, .

[9] S. Rifai, P. Vincent, X. Muller, X. Glorot and Y. Bengio, *Contractive auto-encoders: Explicit invariance during feature extraction*, .

[10] S. Rifai, Y. N. Dauphin, P. Vincent, Y. Bengio and X. Muller, *The Manifold Tangent Classifier*. Curran Associates, Inc., 2011.

[11] S. Rifai, Y. Bengio, Y. Dauphin and P. Vincent, *A generative process for sampling contractive auto-encoders*, `1206.6434`.

[12] G. Puppo, *Kinetic models of bgk type and their numerical integration*, `1902.08311`.

[13] K. Hornik, M. Stinchcombe and H. White, *Multilayer feedforward networks are universal approximators*, *Neural Networks* **2** (Jan., 1989) 359–366.

[14] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, `1603.07285`.

[15] J. Reiss, *Skript zu cfd 1*, `1603.07285`.

[16] H. L. Dret and B. Lucquin, *Partial Differential Equations: Modeling, Analysis and Numerical Approximation* -. Birkhäuser, Basel, 2016.

Acknowledgement

Hilfsmittel

.

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Seitens des Verfassers bestehen keine Einwände, die vorliegende Masterarbeit für die öffentliche Benutzung im Universitätsarchiv zur Verfügung zu stellen.

Berlin, den 15. Mai 2017

Philipp Krah