Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science

Zachary Schellin
376930

January 7, 2021

Contents

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 State of the art

State of the art model reduction of dynamical systems is 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 lower rank.

$$f(x) \approx \tilde{f}(x) \qquad \text{with} \qquad rk(f(x)) \gg rk(\tilde{f}(x)) \tag{1}$$

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 1 % [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 Ja-

cobian of the bottleneck layer with respect to the input $\vec{x}_m = JJ^T$, sampling is realized [11].

1.2 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. 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.....

1.3 Thesis outline

2 The BGK Equation

The Knudsen number eq. (2) introduced by Danish physicist Martin Knudsen is a measure for the rarefaction of gases. In eq. (2) λ represents the mean free path and L the characteristic length [4] of a particle. The mean free path describes the average distance a particle may travel between successive impacts [**WIKI**]. For idealized gases the mean free path can be calculated via eq. (3). In eq. (3) k_b is the Boltzman constant, p is the total pressure, T is the thermodynamic temperature and d is the hard shell diameter [**WIKI**].

$$Kn = \frac{\lambda}{L} \qquad (2) \qquad \lambda = \frac{k_b T}{\sqrt{2}\pi d^2 p} \qquad (3)$$

For Kndusen numbers $Kn > 10^{-2}$ collisions are predominant in comparison to free transport, whereas for $Kn < 10^{-2}$ free transport becomes the predominant behavior compared to collision[4]. This difference in turn characterizes flows where the Bolzmann equation (collisions) or the Navier-Stokes equations (free transport) are valid. Hence the former eq. (4) describes the dynamics of a gas flow, where f is the probability density distribution function for a particle at point $\mathbf{x} \in \mathbb{R}^3$ with velocity $\xi \in \mathbb{R}^3$ at time $t \in \mathbb{R}$.

$$\partial_t f(\mathbf{x}, \xi, t) + \xi \Delta_x f(\mathbf{x}, \xi, t) = Q(f, f) \qquad (4)$$

Originally Q is often the binary Boltzmann collision term, which can be intractable in pratice [1]. Thus the BGK equation utilizes the BGK-Operator for the collision term Q(f,f) eq. (5)[4].

$$Q(f, f) = \frac{M_f(\mathbf{x}, \xi, t) - f(\mathbf{x}, \xi, t)}{\tau(\mathbf{x}, t)} \qquad (5)$$

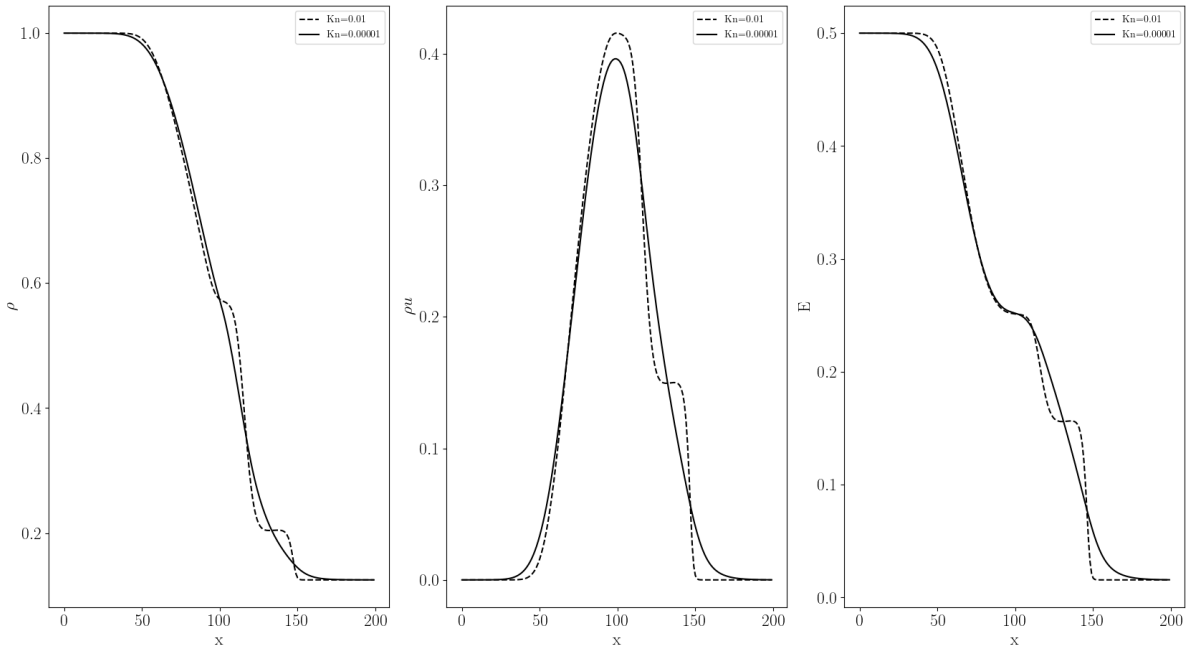
$$M_f(\mathbf{x}, \xi, t) = \frac{\rho(\mathbf{x}, t)}{(2\pi T(\mathbf{x}, t))^{\frac{3}{2}}} \exp(-\frac{||\xi - U(\mathbf{x}, t)||^2}{2T(\mathbf{x}, t)}) \qquad (6)$$

$$\tau(\mathbf{x},\mathbf{t}) = \frac{Kn}{\rho(\mathbf{x}, t) T^{1-\nu}(\mathbf{x}, t)} \qquad (7)$$

For many kinetic gas problems it is sufficient to replace the complexity of the Boltzmann collision term by a mean-free-path approach. τ is referred to as the relaxation time, considering the fact, that collisions tend to relax the distribution function to an equilibrium state f_0, where $f_0 = M_f$ the equilibrium state is approximated by a Maxwellian distribution function[1]. In eq. (6) $U(\mathbf{x}, t) = (u(\mathbf{x}, t), v(\mathbf{x}, t), w(\mathbf{x}, t))^T$ is the macroscopic velovity, $T(\mathbf{x}, t) \in \mathbb{R}$ and $\rho(\mathbf{x}, t) \in \mathbb{R}$is the temperature and density of the gas respectively. In eq. (7) $\nu \in \mathbb{R}$ is the exponent of the viscosity law of the gas. As a result one obtains th BGK equation which can be utilized for both hydrodynamic and rarefied regimes and meets global conservation as discussed in section 2.1. By multiplying the BGK equation with the collision invariants $\Phi(v) = (1, \xi, \frac{1}{2}\xi^2)^T$ and integrating over velocity space $d\xi$, one obtains the corrsponsing moments, eq. (8) to eq. (10).

$$\rho(x,t) = \int f d\xi \quad (8) \qquad \rho u(x,t) = \int \xi f d\xi \quad (9) \qquad E(x,t) = \int \frac{1}{2}\xi^2 f d\xi \quad (10)$$

2.1 Macroscopic features of Hydrodynamic and Rarefied Gas Flows in the SOD-Shock Tube



- intrinsic code variables for hydro an rare add pls

3 Deep learning

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 $f^\star(\mathbf{x};\mathbf{\Theta}) \approx f(\mathbf{x})$ through learning the values of the parameters $\mathbf{\Theta}$. As mentioned before $f^\star$ is a composition of functions eg. eq. (11).

$$f^\star(\mathbf{x};\mathbf{\Theta}) = f^3(f^2(f^1(\mathbf{x};\mathbf{\Theta}^1),\mathbf{\Theta}^2),\mathbf{\Theta}^3) \tag{11}$$

In eq. (11) 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 [12] 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. (12), 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.

$$y = xA^T + b \tag{12}$$

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. (13), `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 13 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. (14) x is represented by $I(m, n)$, and w by $K(m, n)$ respectively. Note that while eq. (13) is scalar valued and eq. (14) is two dimensional, only for illustrative reasons.

$$s(t) = (x * w)(t) = \int x(a)w(t - a)\, da \tag{13}$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{14}$$

One drawback in implementing the discrete convolution eq. (14) 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. (15). Without the need of flipping the kernel which is not always possible, i.e. exploiting the commutative property, cross correlation is adopted eq. (16).

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{15}$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \tag{16}$$

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

$$o = \frac{i - k}{s} + 1. \tag{17}$$

Here o, i, s and k are the output, input, stride and kernel size of one dimension respectively [13]. 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. (16) yields

$$\mathsf{S}_{i,j,k} = c(\mathsf{K}, \mathsf{I}, s)_{i,j,k} \sum_{l,m,n} \left[\mathsf{I}_{l,(j-1)\times s+m,(k-1)\times s+n} \mathsf{K}_{i,l,m,n}\right]. \tag{18}$$

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. (18) 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 [13]. Note that the padding of the kernel with zeros is only used to illustrate how the upsampling works. In eq. (19) taken from [6] multiplications with zero are omitted. The size of one dimension during upsampling can be calculated with the transposition of eq. (17).

$$t(\mathsf{K}, \mathsf{H}, s)_{i,j,k} = \sum_{\substack{l,m \\ s.t \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ s.t \\ (n-1)\times s+p=k}} \sum_{q} \mathsf{K}_{q,i,m,p}\mathsf{H}_{q,l,n} \tag{19}$$

Forward-propagation is then referred to as the compositional evaluation of each layer. For the example in eq. (11) 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. (11) back-propagation would be equations eq. (22) to eq. (20):

$$\frac{\partial J}{\partial \mathbf{\Theta}^3} = \frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}}{\partial \Theta^3} \tag{20}$$

$$\frac{\partial J}{\partial \Theta^2} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial q}\frac{\partial q}{\partial \Theta^2} \tag{21}$$

$$\frac{\partial J}{\partial \mathbf{\Theta}^1} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial \Theta^1} \tag{22}$$

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

4 Reduced Order Algorithms

4.1 Data Sampling

DIE ORIGINAL DTATENSTRUJTUR BESCHREIBEN For the autencoder using fully connected layers, the input vectors $y_o \in \mathbb{R}$ of size $n_{input} = n_\xi = 40$ are arranged in the sampling matrix $S_{AE} \in \mathbb{R}^{5000x40}$ as seen in eq. (23) 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

$$S_{AE} = \begin{bmatrix} f(\xi_1, t_1, x_1) & \cdots & f(\xi_n, t_1, x_1) \\ f(\xi_1, t_1, x_2) & \cdots & f(\xi_n, t_1, x_2) \\ \vdots & \vdots & \vdots \\ f(\xi_1, t_1, x_n) & \cdots & f(\xi_n, t_1, x_n) \\ f(\xi_1, t_2, x_1) & \cdots & f(\xi_n, t_2, x_1) \\ \vdots & \vdots & \vdots \\ f(\xi_1, t_n, x_n) & \cdots & f(\xi_n, t_n, x_n) \end{bmatrix} \tag{23}$$

$$S_{Conv} = \begin{bmatrix} n_{Filters} & f(\xi_1, \mathbf{t}, \mathbf{x}) \\ n_{Filters} & f(\xi_2, \mathbf{t}, \mathbf{x}) \\ \vdots & \\ n_{Filters} & f(\xi_n, \mathbf{t}, \mathbf{x}) \end{bmatrix} \tag{24}$$

Convolutional autoencoders use a different sampling matrix S_{Conv} due to their two dimensional capability resulting in $n_S = 40$ available samples eq. (24).$N_{Filters}$ varies over the succeeding layers, growing with the shrinkage of $(\mathbf{t}, \mathbf{x})$.

4.2 POD

The singular value decomposition of the input X [REF to Section 1] gives the optimal low-rank approximation $\tilde{X}$ of X eq. (25)[Eckard-Young].

$$\underset{\tilde{X}, s.t. rank(\tilde{X})=r}{\operatorname{argmin}} \quad ||X - \tilde{X}||_F = \tilde{U}\tilde{\Sigma}\tilde{V}^* \tag{25}$$

4.3 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 hydrodynamic and rarefied input data.

4.3.1 Hyperparameters for the Fully Connected Autoencoder

To start with a working model an educated guess is made about the initial design of the architecture. **??** shows chosen hyperparameters which are used. Originally the number

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

Table 1: Initial hyperparameter selection

of layers is determined, as they set a 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 fig. 10 in section 5.4 show the training and validation error for five designs shown in table 2.

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

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 2: Initial hyperparameter selection

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.
In the following the width of the layers will be analyzed, by varying two parameters. The hidden dimension and the intrinsic dimension. There are two available layers for the encoder(the architecture of the decoder mirrors that of the encoder) from which the input layer shrinks the input dimension to the hidden dimension and subsequently the bottleneck layer shrinks the hidden dimension to the intrinsic dimension. For the hydrodynamic regime the size of the intrinsic dimension is set, as described in section 2 to three. Therefore only the hidden dimension has to be found. The results of five experiments are shown in table 3, setting the size to 40 hidden units.

Hidden units	10	20	30	40	50
Error	0.0054	0.0027	0.0036	0.0017	0.0032

Table 3: L2-Error for different number of hidden units for the hydrodynamic regime.

The number of intrinsic variables defining the rarefied regime is unknown. Therefore experiments varying the the number of intrinsic variables from two to ten are performed.

Intrinsic variables	2	3	5	6	7	8	9	10
Error	0.0059	0.0049	0.0026	0.0027	0.0017	0.0022	0.0017	0.0015

Table 4: L2-Error for different numbers of intrinsic variables.

Afterwards the activations in the layers will be studied. First five activations are applied to all the four layers. Second a combination of activation functions is studied, where the

intrinsic variables will be activated with a different function than the remaining layers. After training the L2-Error?? will be evaluated on the unshuffled, complete dataset as described in section 4.1. Results can be observed in table 6. ELU and SiLU stand out

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 5: L2-Error different activation functions and combinations for the hydrodynamic regime.

Activation function	ReLU	ELU	Tanh	SiLU	LeakyReLU
Error	0.0328	0.0178	0.0125	0.0134	0.0183
Activation function	ELU/Tanh	LeakyReLU/Tanh	ELU/SiLU		
Error	0.0019	0.0017			

Table 6: L2-Error different activation functions and combinations for the rarefied regime.

with the lowest loss.

4.3.2 Hyperparameters for the Convolutional Autoencoder

The convolutional autoencoder architecture 1.1 is a composition of six convolutional and three fully conected layers, eq. (26).

$$y_p = f_C^9(f_C^8(f_C^7(f_F^6(f_F^5(f_F^4(f_C^3(f_C^2(f_C^1(y_0)))))))))) \tag{26}$$

4.3.3 Training

During training every 1000 epochs a sample against its prediction was printed in order to link the value of the L1-Loss to a prediction. Using this method a first verification of the model was achieved. Continuing the search for any possible shortage of the models performance, that this method could not cover, eg. samples lying between every 1000 sample, that the model was not able to reconstruct correctly, a second verification process is conducted. Analysing the batch size for the architecture 1.0 the test errors in table 8 can be produced.

4.4 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 fig. 4. Hence the expolitability of the code in terms of constructing a ROM is not provided. On that account the method of characteristics [14] provides a means to bypass this shortage. It is necessary for $c_i(x,t)$ to satisfy the conservative condition eq. (27) and

Kn	0.00001	Kn 0.01
Batch Size	L2-Error	
64	0.008	
32	0.0049	
16	0.0038	
8	0.0037	
4	0.0026	
2	0.0021	

Table 7: L2-Error over Batch-Size

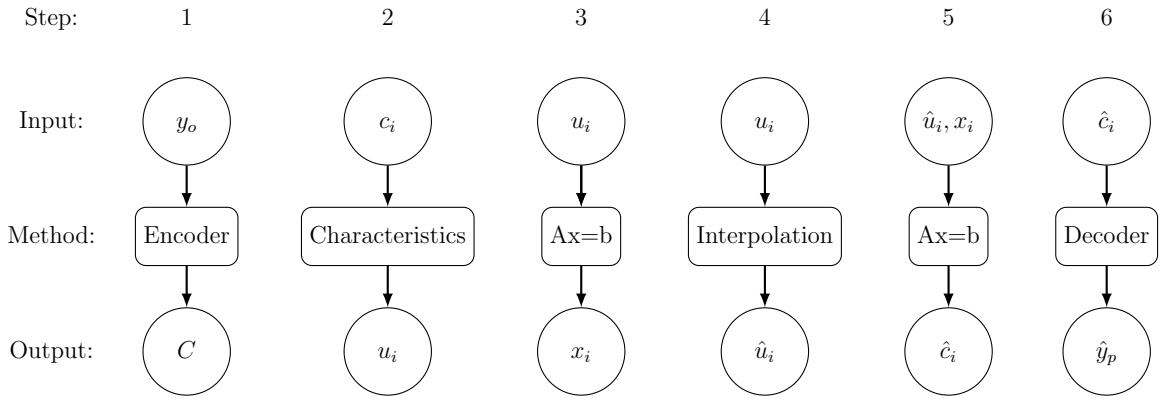


Figure 1: This figure shows the steps for obtaining a reduced oder model (ROM). Decoder and Encoder need to be used after training. In step one y_0 is the original input data, C is the Code. In step two c_i is the i-th intrinsic variable and u_i the correspnding characteristic. The eigenvalue problem 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.

the transport equation eq. (28).

$$\frac{d}{dt}\int c_i \, dx = \frac{d}{dt} f_i = const. \qquad (27) \qquad\qquad \frac{\partial}{\partial t} c_i + \frac{\partial}{\partial x} f_i = 0 \qquad (28)$$

The characteristics u_i describe the constant transport velocities for each variable c_i calculated using eq. (29). 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 1 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 section 5.

$$u_i = \frac{f_i(c_i^-) - f_i(c_i^+)}{c_i^- - c_i^+} \qquad (29)$$

5 Results

5.1 Hydrodynamic Regime

In search for a reduced model of the BGK equation, a first reduction and analysis of the provided data in the hydrodynamic regime is conducted. The error over the L_2-Norm derived from eq. (32) assigns a value to each reduction algorithm enabling a evaluation. Furthermore the conservation quantities given in eq. (8) to eq. (10) of the prediction are

Algorithm	L_2
SVD	0.03
Fully Connected Autoencoder	0.002
Convolutional Autoencoder	0.02

Table 8: L2-Error over Batch-Size

analysed over the time average of each quantity. This normalization is given in eq. (31), where $\hat{\sigma}$ represents the given quantity. With more than 99% of the total cumultative energy S_N of the first five singular values calculated with eq. (30) the SVD provides an upper bound to the number of intrinsic features the autoencoder should extract. Figure 2 shows the singular values (left) and the cumulative energy (right).

$$S_N = \sum_{k=1}^{N} a_k \qquad \text{with a sequence} \qquad \{a_k\}_{k=1}^{n} \tag{30}$$

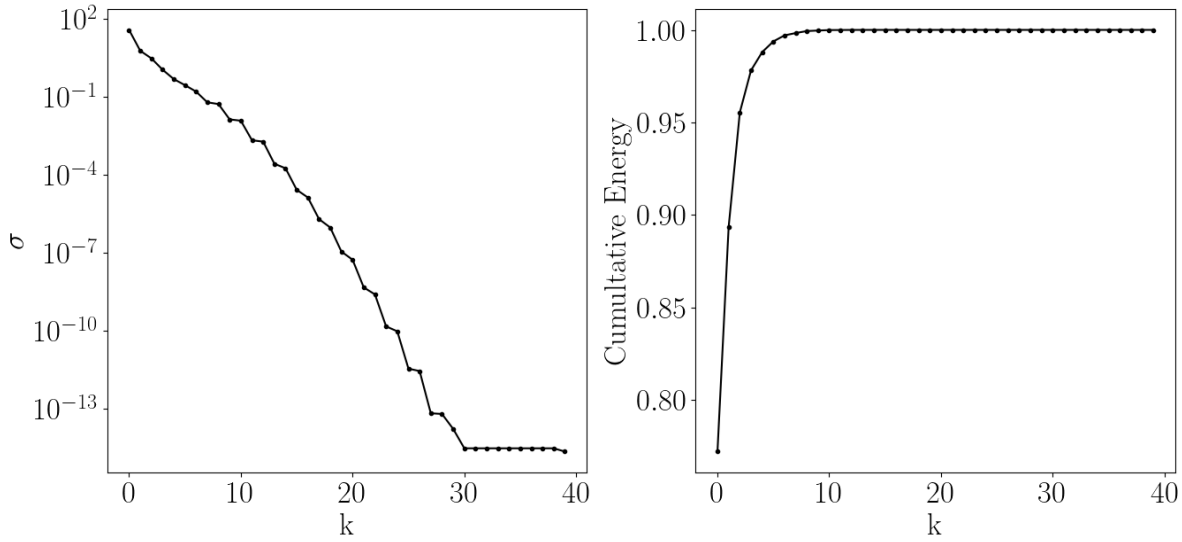


Figure 2: Singular Values (left) and cumultative enrgy (right) over the number of singular values

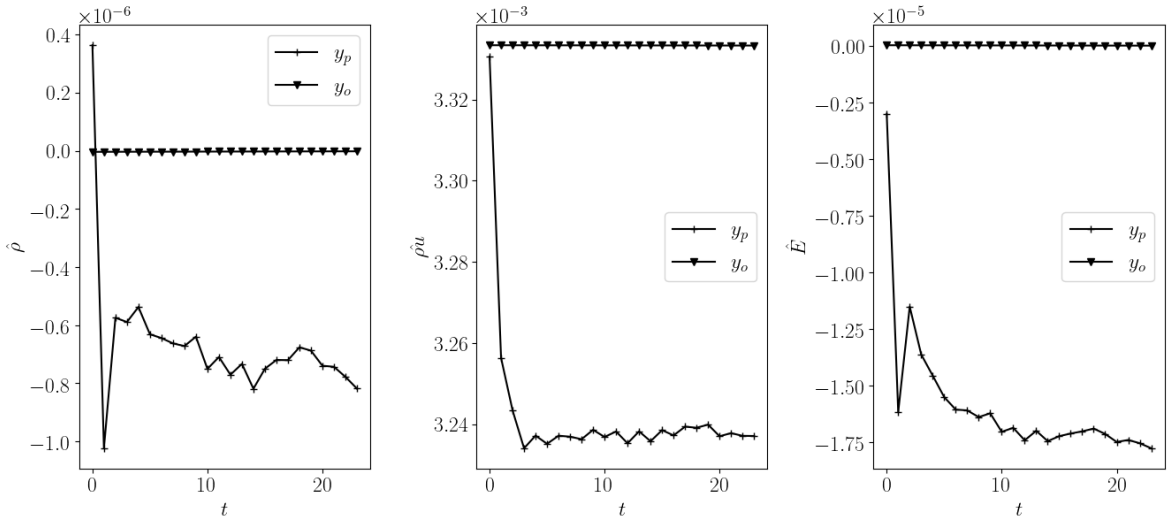
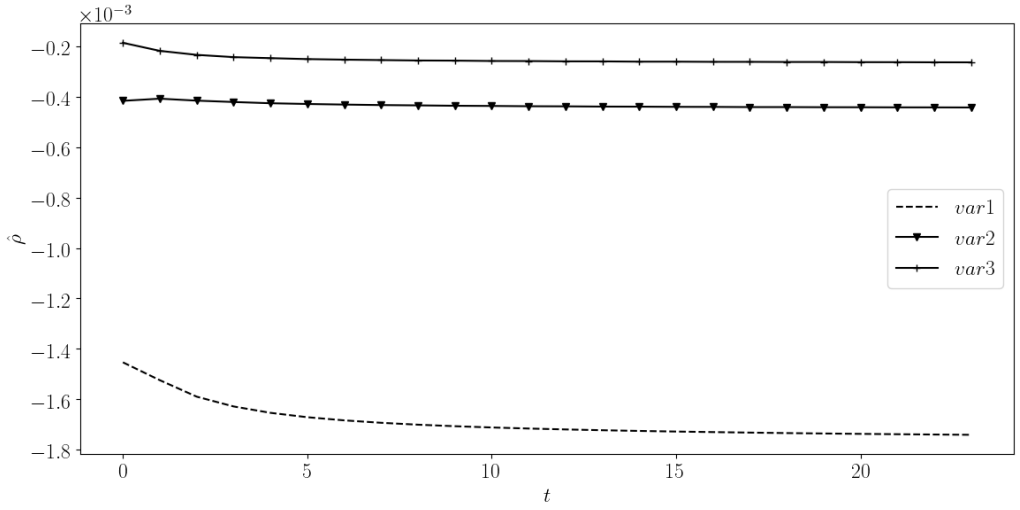


Figure 3: Normalized conservative quantities $\hat{\rho}$, $\hat{\rho u}$ and $\hat{E}$ as in eq. (31) for y_o and y_p.



$$\hat{\sigma} = \frac{\frac{d}{dt}\int \sigma \, dx}{\bar{\sigma}} = 0 \quad \text{and} \quad \bar{\sigma} = \frac{\iint \sigma \, dt dx}{\Delta t} \tag{31}$$

$$L_2 = \frac{||y_o - y_p||}{|||y_o|} \tag{32}$$

Given, that the autoencoder is able to achieve a reconstruction that is equal or below the threshold of the L_2-Norm form existing methods like SVD **??** and that conservation is preserved, a reduced order model can not be derived as described in **??**. Solely the the reconstuction can be validated. In addition the code needs to be a conservative system hence fulfilling eq. (8) to eq. (10).

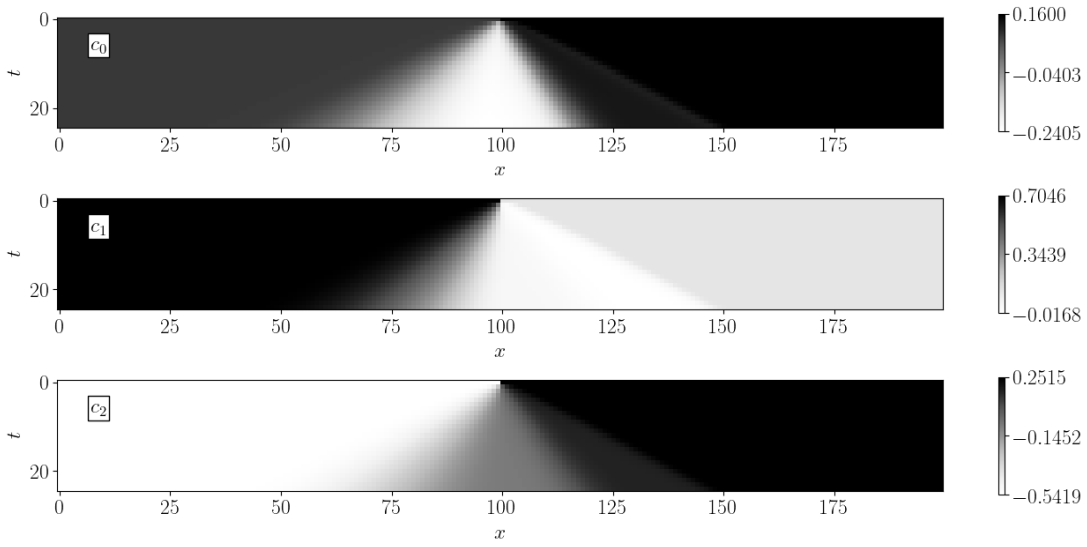


Figure 4: Code variable c_1,c_2 and c_3 over space x and time t of the fully connected autoencoder

5.2 Rarefied Regime

5.3 Discussion and Outlook

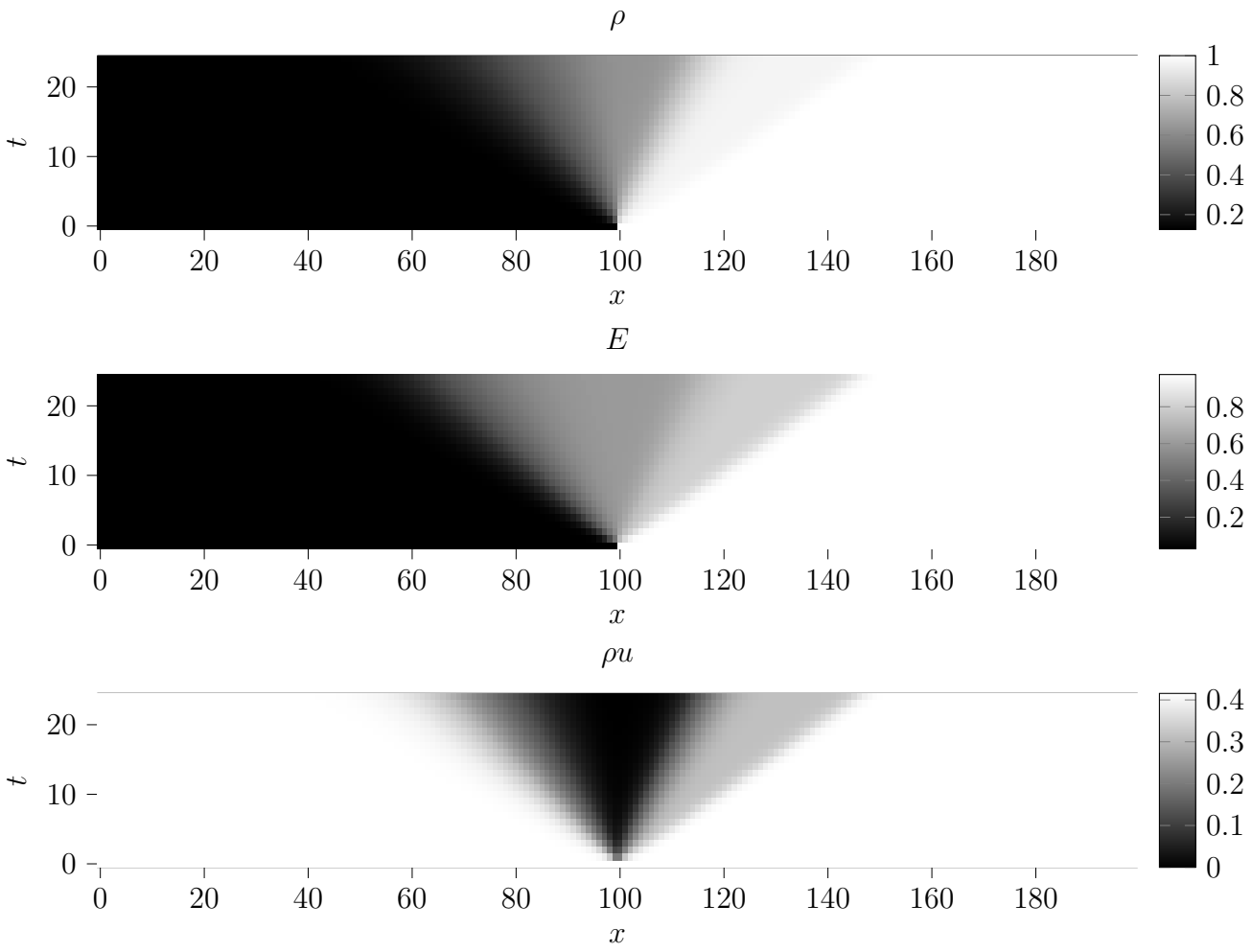


Figure 5: Macroscopic quantities ρ, E and ρ u of the original data.

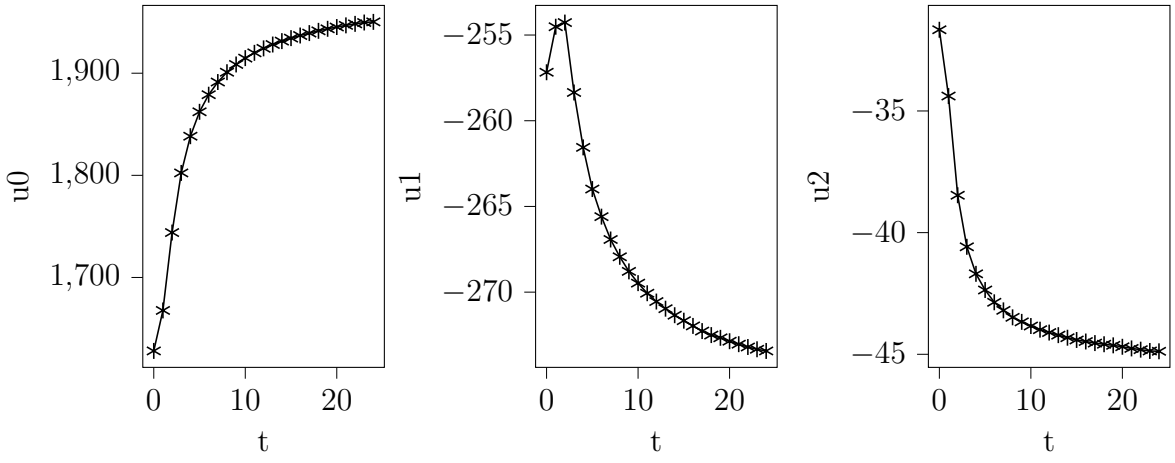


Figure 6: Characteristic velocities u0, u1, u2 of the code variables var0, var1, var2 respectively calculated as described in Section 4.4.

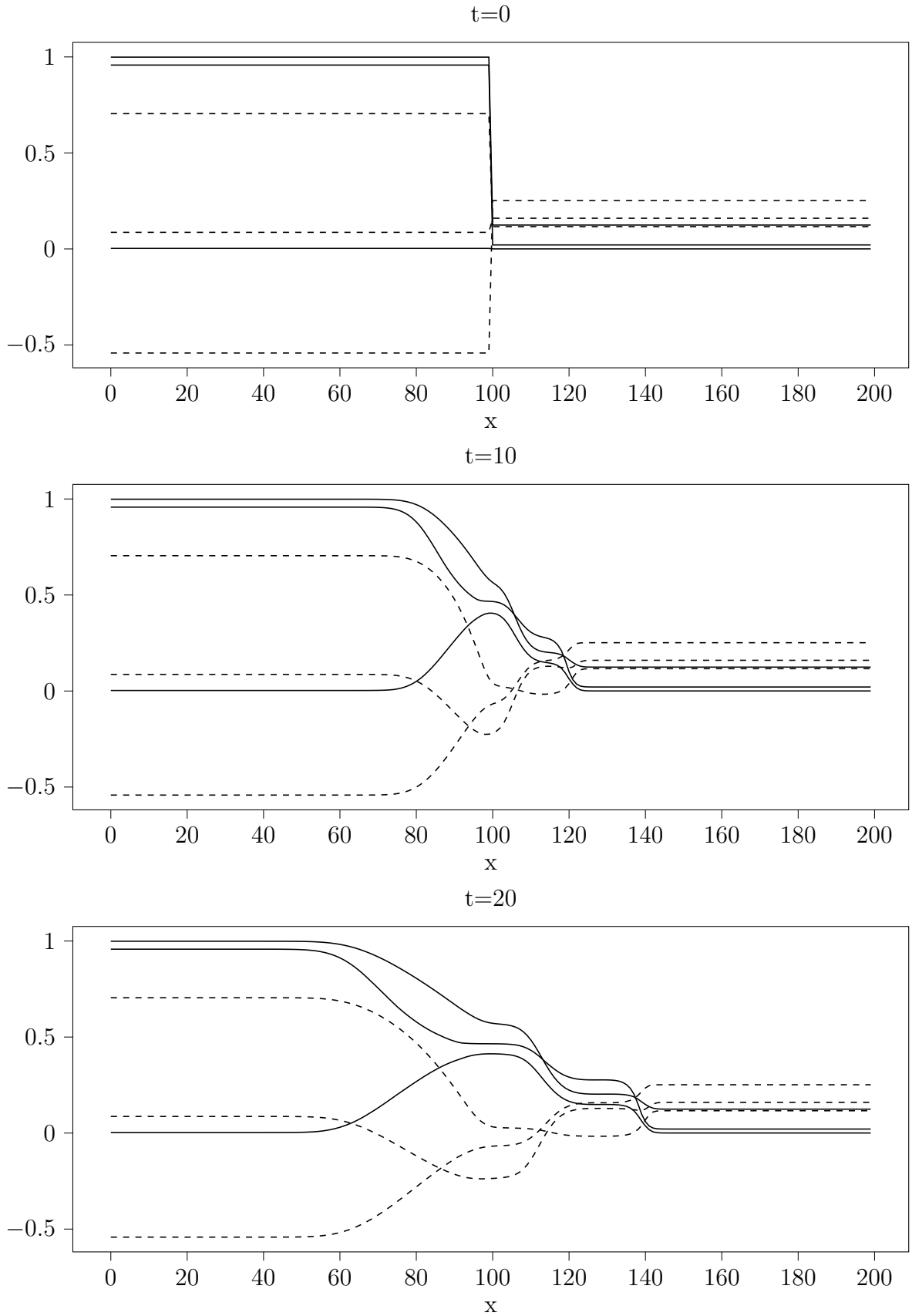


Figure 7: Code variables var0, var1, var3 (dashed lines - -) and macroscopic quantities ρ, E, ρu (full lines –) for three timestamps t_0, t_{10}, t_{20}.

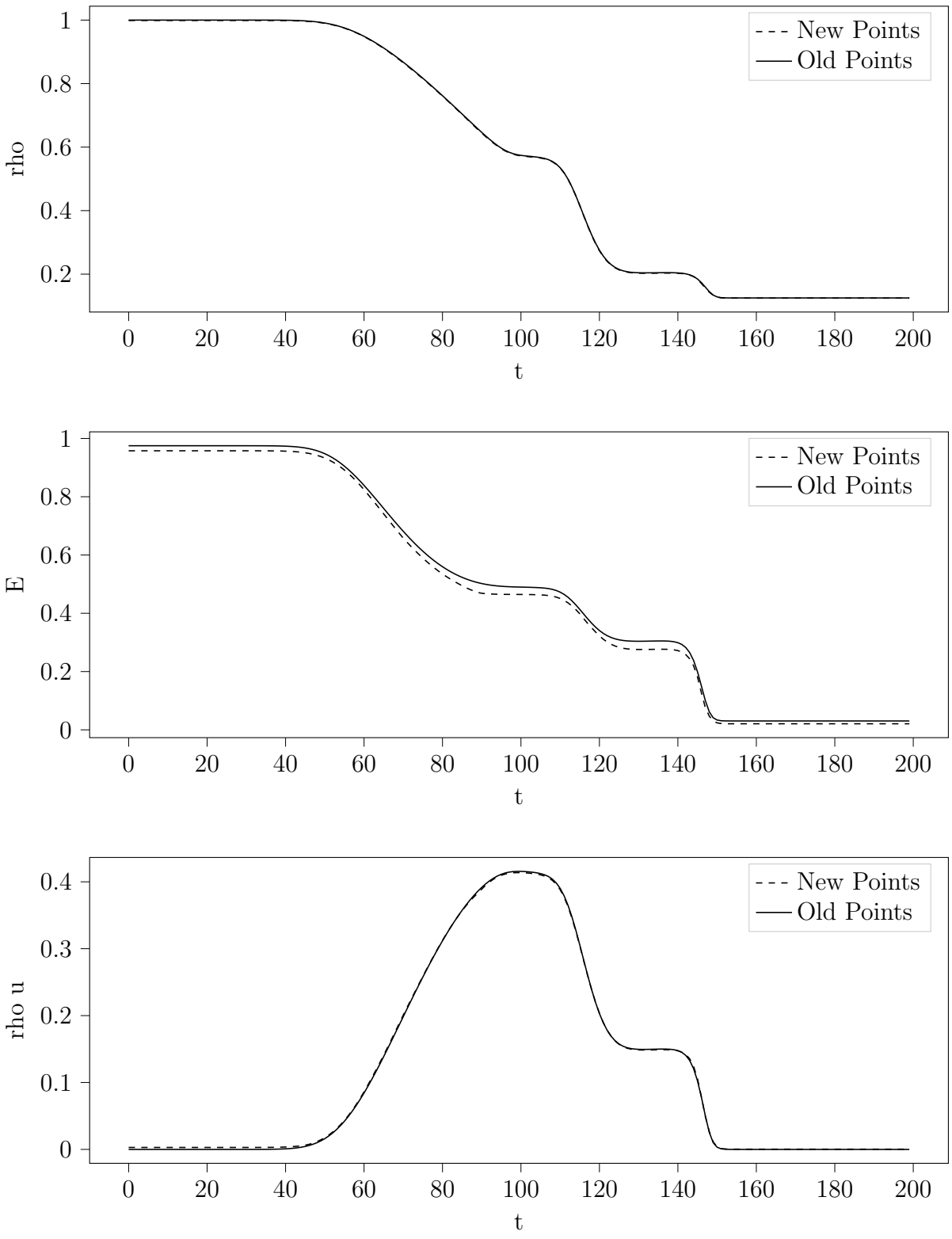


Figure 8: Resulting macroscopic quantities ρ, E, ρu after interpolating in time using the interpolation method described in 4.4. The interpolated quantity lies at timestamp t=24,5, while the original quantity lies at timestamp t=25.

References

[1] Bhatnagar, Gross, and Krook. A model for collision processes in gases. 1954.

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

[3] Steve L. Brunton and J. Nathan Kutz. *Data driven science and engineering*. 2019.

[4] Florian Bernard, Angelo Iollo, and Sebatian Riffaud. Reduced-order model for the bgk equation based on pod and optimal transport. 2018.

[5] Kookjin Lee and Kevin T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. 2019.

[6] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[7] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. 1986.

[8] Dana H. Ballard. Modular learning in neural networks. 1987.

[9] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. 2011.

[10] Salah Rifai, Yann N Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. *The Manifold Tangent Classifier*. Curran Associates, Inc., 2011.

[11] Salah Rifai, Yoshua Bengio, Yann Dauphin, and Pascal Vincent. A generative process for sampling contractive auto-encoders. 2012.

[12] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989.

[13] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. 2018.

[14] Hervé Le Dret and Brigitte Lucquin. *Partial Differential Equations: Modeling, Analysis and Numerical Approximation* -. Birkhäuser, Basel, 2016.

5.4 Appendix A

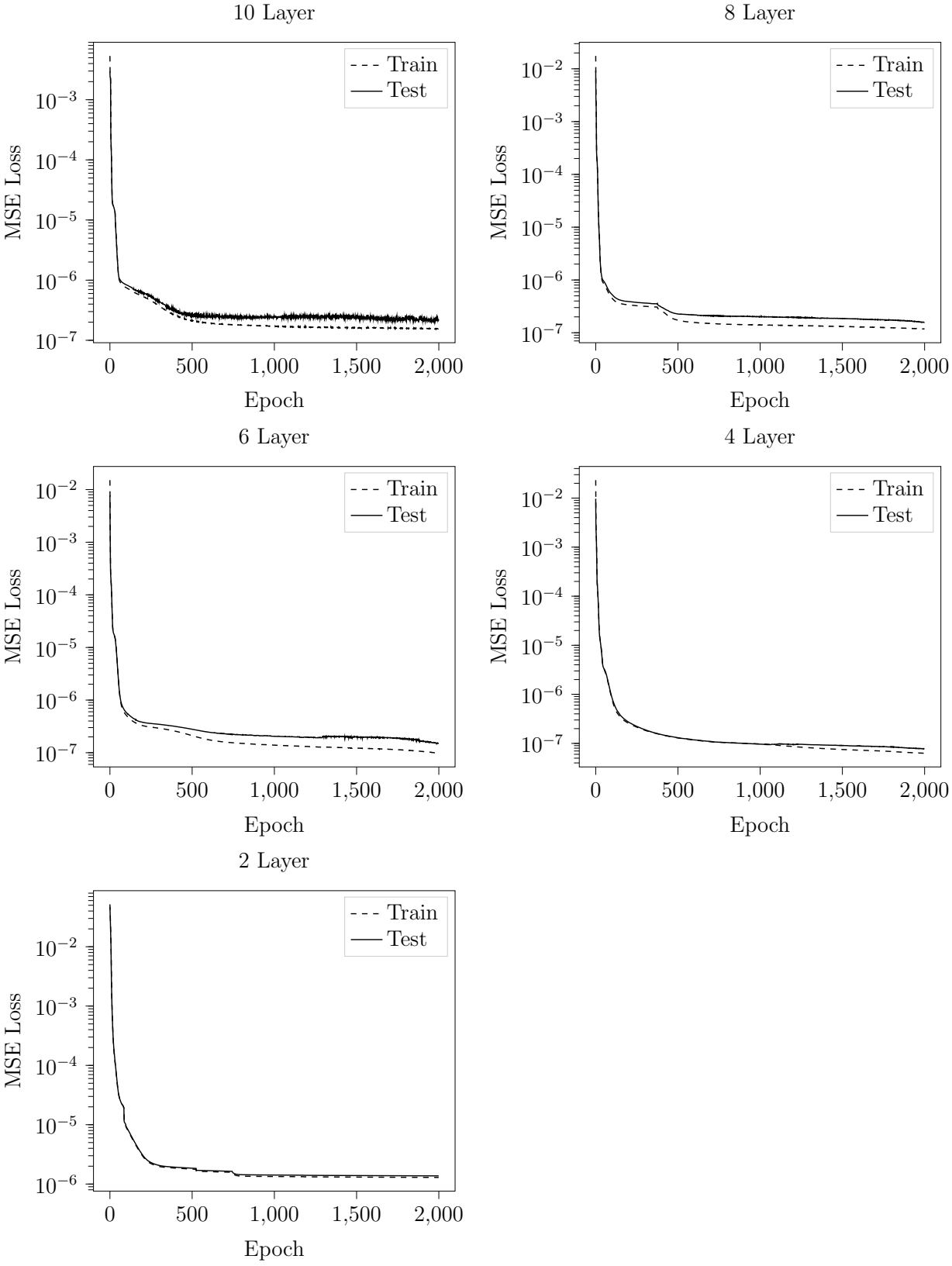


Figure 9: Analysis of layer size for five different architectures, showing the error for training and testing on the data in the hydrodynamic regime.

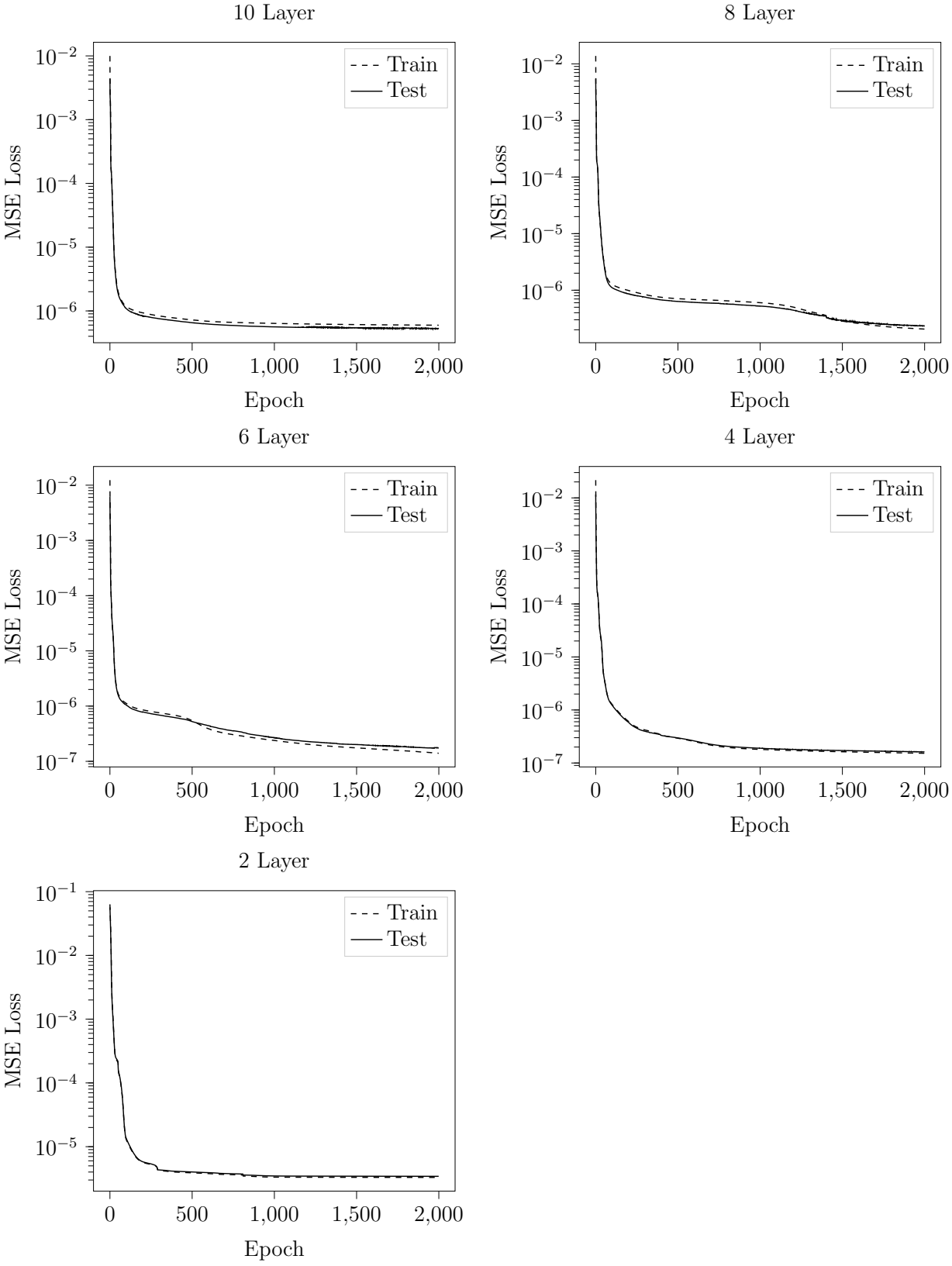


Figure 10: Analysis of layer size for five different architectures, showing the error for training and testingon the data on the rarefied regime.