Unsupervised Learning of Parametric Optimization Problems

1th Thomas Jan Kalekiewicz
Matr.: 202520

2th Anil Can Karsli
Matr.: 235052

3th ZubairAhmed Ansari
Matr.: 243082

***Abstract*—This paper investigates a parametric optimization problem with implementing supervised and unsupervised learning method and in the end comparing both. A major goal in the unsupervised learning is to obtain data representation without knowing labeled data in the training. To achieve this, a neural network structure is modeled using mini-batch stochastic gradient descent and a custom loss function is formulated. The model performance is quite consisted with the supervised approach. Overall, the predictions obtained with both approaches are accurate and in accordance with validation data.**

I. Introduction

Supervised learning is a powerful tool if high amount of labeled data is available. A common understanding suggests that unsupervised method will yield similar powerful results when labels are difficult, impractical to collect or prediction targets are unknown. One popular example to this field is model predictive control where optimal inputs are obtained by solving optimization problem. However, unsupervised learning has yet to fulfill this task. One explanation for this failure is that unsupervised learning algorithms mismatched with target tasks. Many unsupervised objectives find optimum for generative model to produce useful results with only as a side effect [1,2]. In parametric optimization, optimization problem can be solved with representing it as a group of problems rather than single one. These problems are generated by parameters, such that each combination is related to a specific optimization problem. The solution of these problems is a mapping of parameters with each associated problems [3].When the dimensions of parameter increases, algorithms deal with a large number of optimizations in order to estimate the solution accurately. Eventually, this leads to intractable results. As result existing algorithms work poorly. To overcome this difficulty neural networks can be implemented[3]. This paper demonstrates how neural networks (NN) are implemented to solve large-dimensional parametric optimizations instead of using a solver.

II. Methods

A. Optimization problem

In this paper, the following non-linear convex problem is considered to investigate the proposed idea.

$$\min_{xyz} \; = \; \frac{1}{xyz} \tag{1}$$

$$st. \; (xy \; + \; yz \; + \; xz) \; - \; a \; \leq \; 0$$

$$y^b \; - \; x \; \leq \; 0$$

$$x \; > \; 0$$

$$y \; > \; 0$$

$$z \; > \; 0$$

for all a $\in \; R_{++} \; and \; b \; \in \; R$

The problem has only 'a' and 'b' parameters as degrees of freedom. The data for the task is generated by solving the problem in python with cvxpy library. For simplicity parameter range of [5,10] is investigated. The contour plot of the solver results is shown in the figure below.

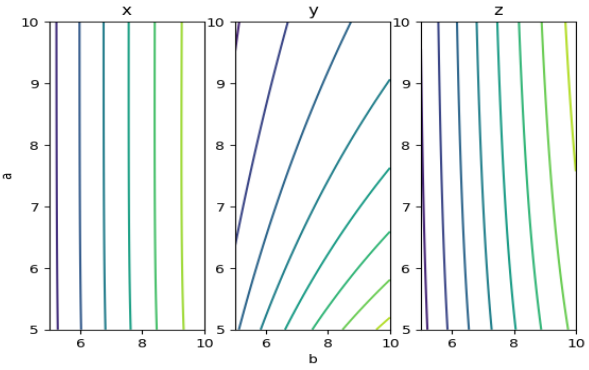

Fig. 1. Contour plot of the solver solution.

B. Data Generation

Training and validation data of the parameter inputs and their corresponding solutions of x,y,z are needed. The parameter room for a and b each having values within the interval 5 and 10 should be covered well. To facilitate this the a and b intervals, which are the same, are split into 100 parts. Afterwards a meshgrid is created and each point is a solution of the given optimization problem for a given parameter combination. So the created meshgrid is solved columwise, wheras the corresponding solutions of x,y,z are saved in lists that are concatenated in the following. The different a and b combinations are concatenated and saved in a list as a tuple, so that there is a large dataset of parameter inputs and the solutions of x,y,z as an output-triple. Each row of the input

list corresponds then to the same row in the output list. In order to streamline the training the lists are split into training and validation data at a ratio of 3:1. These data were saved to have the same data base for further experiments and trainings

C. Toolboxes and Systems

To create and prepare the data for the NN functions from numpy and scikit-library were chosen [4,5]. In this project for the given parametric optimization problem solutions with a supervised NN and with an unsupervised NN were found an compared afterwards. Both NN were build using tensorflow and keras functions [6,7].

D. Supervised Learning

The NN is created for estimating the solution of problem in Eq.(1). The proposed NN for supervised approach can be seen in table 1 and is influenced from some other investigations [8]. NN's predictions and labeled data are passing through the mean squared error (MSE) as a loss function, so that MSE is minimized. Lastly, neural network is optimized using Adam optimizer. The Adam optimizer uses mini-batch stochastic gradient descent and adjusts the models parameters in each iteration loop [8,9]. The number of batches is 50. The structure is basically similar to the one for the unsupervised approach, but the ab parameter input refers only to the NN and not to the loss function. Therefore labeled data is taken into account at the loss function (Fig. A1). To prevent overfitting an L2-norm is implemented in each layer. The structure of the supervised learning loop can mainly be implemented with keras toolboxes. After defining the layers with respect to table 1 the NN is compiled with adam optimizer and MSE as loss and metric. In the next step the ab parameter training data and the xyz labels are used to adjust the NN's parameters in a way that the MSE between the supervised NN's xyz predictions and the xyz labels is minimized. The unsupervised NN is trained for 50 epochs.

TABLE I
STRUCTURE OF SUPERVISED NN

layer	Neurons	Activation function	Regularization
Input layer	900	ReLu	L2 norm (λ = 0.001)
hidden layer1	900	ReLu	L2 norm (λ = 0.001)
hidden layer2	900	ReLu	L2 norm (λ = 0.001)
output layer	3	linear	L2 norm (λ = 0.001)

E. Reformulation of optimization problem

One approach for unsupervised learning is to minimize a custom loss function with adjusting the NN's paramters within each epoch. This custom loss function consists of the objective function and weighted constraint terms, which are added to the objective function and can be seen as penalty terms when constraints are violated [8]. In case of the previously mentioned optimization problem the custom loss function can look same as the following equation.

$$L(\widehat{x},\widehat{y},\widehat{z},a,b) = \frac{1}{xyz} + ReLu(xy+xz+yz-a)^3 + ReLu(y^b-x)^3 \tag{2}$$

The very first term is considered as the objective function and the ReLu terms are constraints hardwired in the loss function as penalty terms. In fact this custom loss function allows the penalty terms to vanish if they are fullfilled, because of the ReLu function and in case they are not fullfilled the corresponding ReLu function is potentiated and added. The problem reformulated problem is in unconstrained form.

F. Unsupervised Learning

The Unsupervised NN works in contrast to the supervised approach completely without labeled data and instead tries adjust it's parameters with minimizing a loss function within each epoch [10]. The structure of the unsupervised NN can be seen in table II. The unsupervised NN conists of an input layer and three hidden layers each containing 50 neurons. The activation function in the hidden and the input layer is the ReLu function. The output layer consists of 3 neurons and uses an exponential function as activation function. In that way the NN's outputs regarding x,y,z remain positive and do not violate the constraints. 22

TABLE II
STRUCTURE OF UNSUPERVISED NN

layer	Neurons	Activation function	Regularization
Input layer	50	ReLu	L1 norm (λ = 0.001)
hidden layer1	50	ReLu	L1 norm (λ = 0.001)
hidden layer2	50	ReLu	L1 norm (λ = 0.001)
hidden layer3	50	ReLu	L1 norm (λ = 0.001)
output layer	3	Exponential	L1 norm (λ = 0.001)

The basic principle of the training loop is depicted in (Fig. A2). To prevent overfitting a L1-norm is added on the different layers to the custom loss function [8]. To make training more efficient mini-batch stochastic gradient descent is used with a batch size of 50. Therefore the training input of ab tuples is chunked in 50 batches each containing 150 samples. In each epoch a new batch is chosen and the corresponding loss of the custom loss function is computed. With this value the gradient of the loss with respect to all weights and biases (trainable parameters) is calculated for the entire batch and afterwards forwarded to the adam optimizer [9]. With this information the adam optimizer updates the NN's trainable parameters in each epoch, so that loss is decreasing and the NN's predictions become better [10]. For representing the evolution of the loss value in each epoch a mean metrics is defined.

III. RESULTS

A. Supervised Learning

The performance of the supervised NN is shown in figure 2 and this figure is obtained with the standard setting from

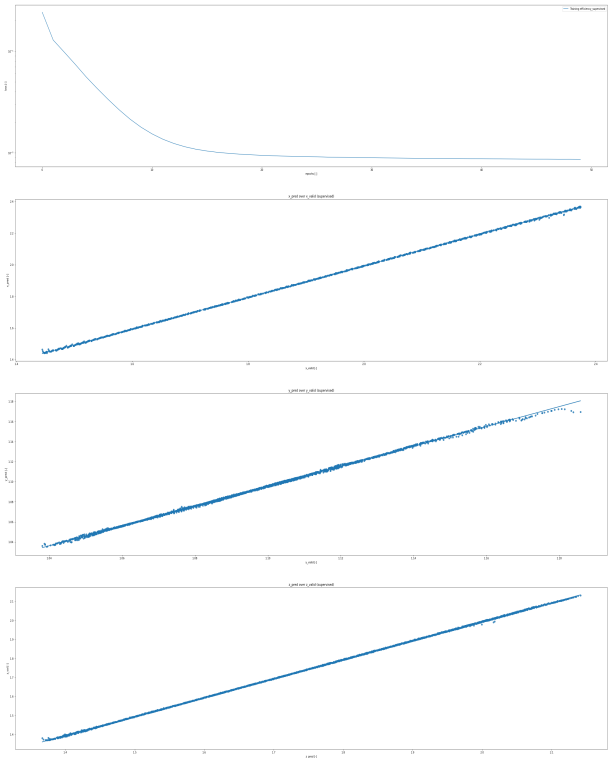

Fig. 2. supervised learning-in the first figure at the top the loss over the epochs is depicted. The lower figures show respectively the predicted x,y,z values over the corresponding validation values.

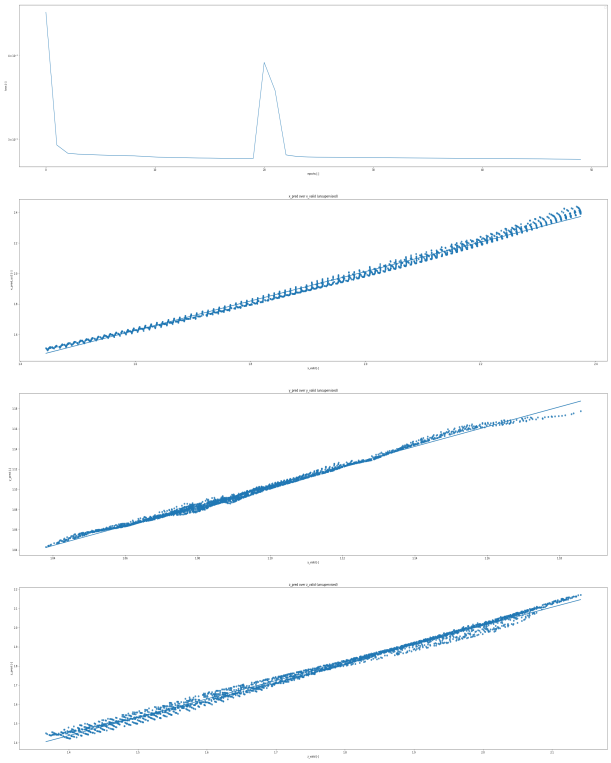

Fig. 3. unsupervised learning-in the first figure at the top the loss over the epochs is depicted. The lower figures show respectively the predicted x,y,z values over the corresponding validation values.

table 1. The graph at the very top of figure 2 depicts the loss over the number of epochs and it can be seen that the loss is monotonously decreasing without anomalies like in the unsupervised learning (figure 3). One notable phenomenon worth mentioning here is that minimizing total error only in training loop does not necessarily yield good results. Overfitting must be considered too, since a good learning algorithm must have good generalizability. One parameter to assess the model performance is the mean squared error (MSE) between the predicted values of x,y,z and the validation data for x,y,z. In this work it was found to be 3.58e-5 which fits to the the three graphs in figure 2, in which the predicted NN values for x,y,z are plotted respectively over the validation values of x,y,z from the validation dataset obtained by the solver. It can be seen that the datapoints lie nearly on a line and therefore there seems to be a linear correlation between the predicted x,y,z values of the supervised NN and the validation x,y,z data and the NN seems to make good predictions and no overfitting occured.

B. Hyperparameter Tuning - Supervised

Modern supervised machine learning algorithms involve hyperparameters that have to be set before running them. Options for setting hyperparameters are default values from the software package, manual configuration by the user or configuring them for optimal predictive performance by a tuning procedure. Three hyperparameters are considered to tune in this work as units (number of Neurons), number of layers (controls depth of the model) and the learning rate (controls how quickly the model learns from data). Since Keras models have their own Interface and are not directly compatible with scikit-learn, KerasRegressor wrapper is implemented to bridge the gap.

TABLE III
HYPERPARAMETERS OBTAINED

CV	Learning Rate	Number of Layers	Units	Accuracy
3	0.1	1	64	96.125
5	0.001	1	128	96.346
10	0.001	1	32	95.971

C. Unsupervised Learning

The results of the unsupervised learning are plotted in figure 3. At the very top of this graphic the loss value is plotted over the epoch number. It can be seen that the loss decreases monotonously, but there is also a very steep increase at the 20th epoch around, which is followed by a very steep decrease afterwards. After this anomaly the loss graph decreases again and the loss converges towards 0.28. Compared to the loss graph in figure 2 the loss function seems to decrease faster, as it reaches the 0.28 earlier. But it does not go to smaller values. Chances are the steep increase decrease of the loss anomaly in the unsupervised learning occurs due to the optimizer. In the three figures below the predicted x,y,z values of the NN respectively are plotted over the corresponding validation

x,y,z values. The regression lines fit the data points quite well, since the predicted values deviate around this regression lines in a not too big manner. So there seems to be a linear correlation between the predicted values from the unsupervised NN and their respective validation values. The MSE for the unsupervised learning is 3.57e-5 which means that the unsupervised NN is a bit more imprecise than the supervised NN. In general unsupervised NNs depend to a high extent on their hyperparameters. Therefore the reason for the inaccuracy is very likely to be related to this. For obtaining figure 3 the optimized unsupervised NN obtained via hyperparemeter tuning is used.

D. Hyperparameter Tuning - Unsupervised

For this part the hyperparameter tuning is done with grid search by varying different model parameters, executing the unsupervised training loop and in the end evaluating the efficiency of this hyperparameter combination with MSE. In the end the hyperparameter combination with the smallest MSE is chosen. The corresponding tables can be found in the appendix. Table A1 encompasses different combinations in terms of hidden relu-layers and different number of neurons per layer. The smallest MSE is around 0.0006 and can be obtained with 4 relu layers each containing 50 neurons. The results can vary during each training a bit and this is probably due to random effects during the training. It can be stated that a higher number of neurons increases the epoch time, since more gradients need to be computed, but the accuracy does not increase necessarily or becomes even worse. In table A2 different activation functions are compared and ReLu function seems to be the most appropriate activation function. Regarding table A3 l1 regularization and 50 epochs show the smallest error. The effects of weighting the constraints can be seen in table A4 and it can be inferred that potentiating the constraints is a harder penalty than multiplication with some weighting factors. It produces a very harsh penalty term if some constraints are violated.

E. Unsupervised - Use of slack variables

Slack variables are extra variables that are incorporated into the linear constraints of a linear program in order to convert them from inequalities to equalities. The equality constraints are reformulated with slack variables, given in equations below

$$(xy + xz + yz) - slack1 = a \tag{3}$$

$$yb - slack2 = x \tag{4}$$

The code incorporates inequality constraints and penalty functions to enforce specific conditions on the predicted values. It uses the Adam optimizer to optimize the model's parameters. As seen in Fig 4 above, the code evaluates the models performance by calculating the mean squared error between the predicted values and the actual validation data. Also,it includes visualization plots to showcase the training efficiency and the performance of the predicted values against the validated data.

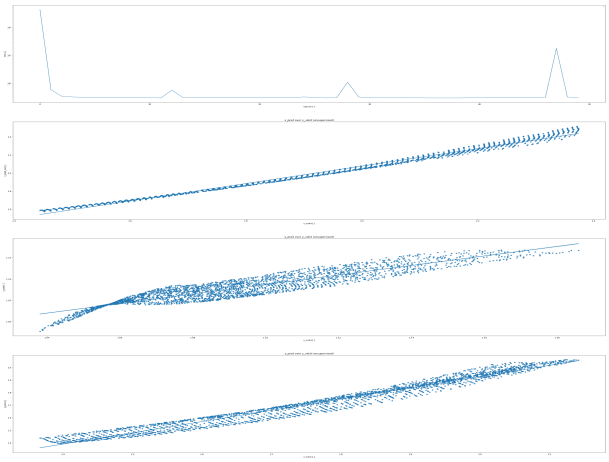

Fig. 4. unsupervised learning(using slack Variables)

IV. Conclusion

In this study neural network implementation in parametric optimization problems is investigated. Two learning approaches of supervised and unsupervised methods are implemented. The key difference between supervised and unsupervised learning is that in unsupervised learning true values of labeled data are not available. Hence, custom loss function is used in unsupervised method. The plot of the combined results can be found in the (Fig. A3). Both learning methods produce reliable results. However, due to the different structure and high sensitivity to hyperparameter tuning unsupervised method generates fuzzy data. In fig.(3), a peak in the training efficiency can be seen. One explanation for this, optimizer first tried to convergence on the local optima and then finds the global minimum. Overall, neural networks with unsupervised learning have promising future in solving parametric optimization problems, but further research can be done in for example in hyperparameter tuning to minimize the loss even more.

section assignment

Thomas Jan Kalenkiewicz: Data Generation, Toolboxes and Systems, Reformulation of optimization problem, Methods-Unsupervised Learning, Results-Unsupervised Learning, Hyperparameter Tuning - Unsupervised
Anil Can Karsli: Abstract, Introduction, Optimization Problem, Methods - Supervised Learning, Results - Supervised Learning, Conclusion
ZubairAhmed Ansari: Hyperparameter Tuning-Supervised, Unsupervised - Use of slack variables

REFERENCES

[1] Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. (2018). Meta-Learning Update Rules for Unsupervised Representation Learning. International Conference on Learning Representations. arXiv preprint arXiv:1804.00222.

[2] Unsupervised learning of parametric optimization problems Project Task Description

[3] R. Nikbakht, A. Jonsson and A. Lozano, "Unsupervised Learning for Parametric Optimization," in IEEE Communications Letters, vol. 25, no. 3, pp. 678-681, March 2021, doi: 10.1109/LCOMM.2020.3027981

[4] "scikit-learn", Accessed: 12.July.2023, https://scikit-learn.org/stable/

[5] "NumPy", Accessed: 12.July.2023, https://numpy.org/

[6] "TensorFlow", Accessed: 12.July.2023, https://www.tensorflow.org/

[7] "Keras", Accessed: 12.July.2023, https://keras.io/

[8] Rasoul Nikbakht, Student Member, IEEE, Anders Jonsson, Angel Lozano, Fellow, IEEE, "Unsupervised Learning for Parametric Optimization", 2021

[9] Trevor Hastie, Robert Tibshirani, Jerome Friedman, "The Elements of Statistical Learning", 2017

[10] Imme Ebert-Uphoff, Ryan Lagerquist, Kyle Hilburn, Yoonjin Lee, Katherine Haynes, Jason Stock, Christina Kumler, Jebb Q. Stewart, "CIRA GUIDE TO CUSTOM LOSS FUNCTIONS FOR NEURAL NETWORKS IN ENVIRONMENTAL SCIENCES - VERSION 1", 2021

[11] "Custom training: walkthrough", Accessed: 12.July.2023, https://www.tensorflow.org

[12] Enas Elgeldawi 1,Awny Sayed ,Ahmed R. Galal and Alaa M. Zaki "Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis" in mdpi/journal/informatics

[13] Philipp Probst, Anne-Laure Boulesteix, Bernd Bischl "Tunability: Importance of Hyperparameters of Machine Learning Algorithms" in Journal of Machine Learning Research 20 (2019) 1-32

[14] "Explanation of Simplex Method" in IMSE Concept Library —Laurel Nichols and Gina Christofaro — Spring 2015

Appendix

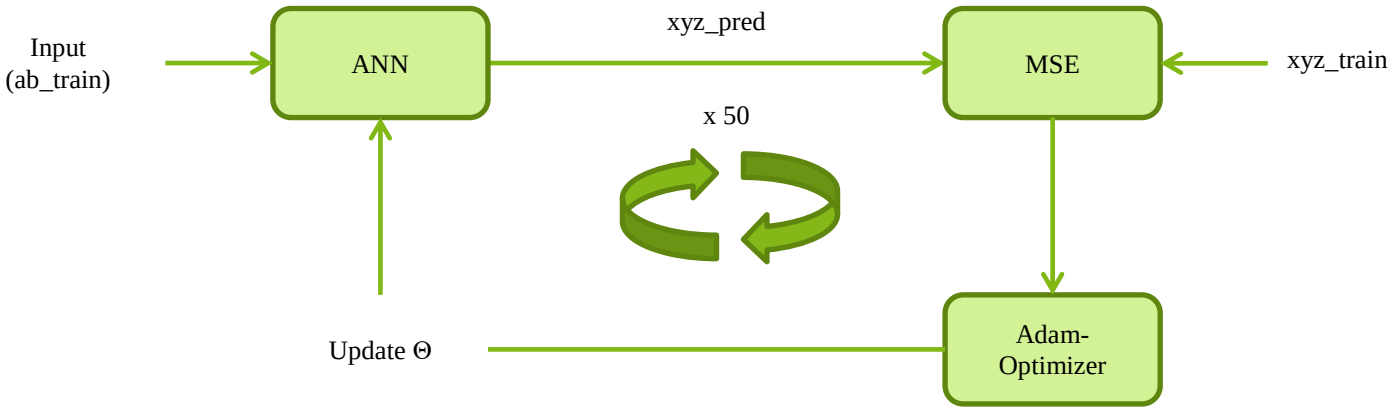

Figure A1: supervised training loop

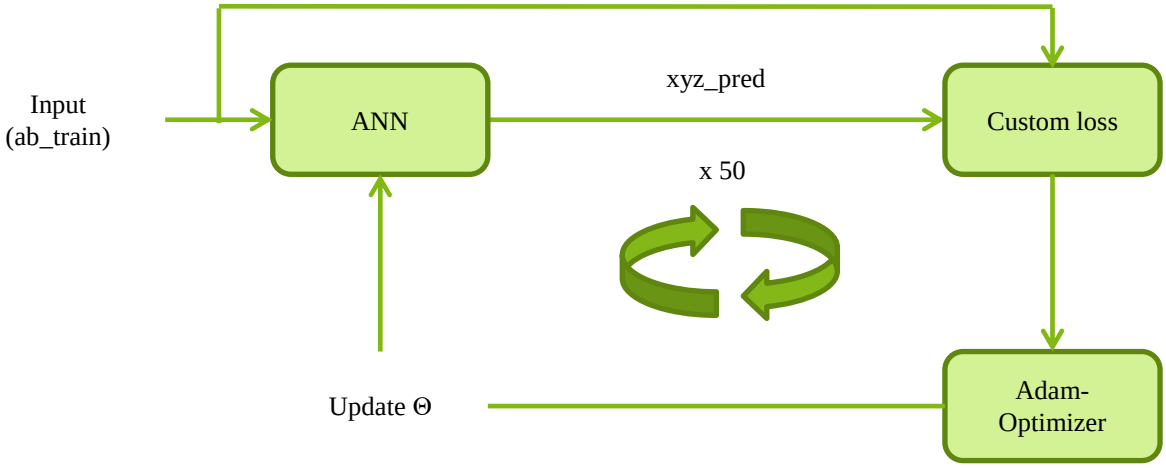

Figure A2: unsupervised training loop

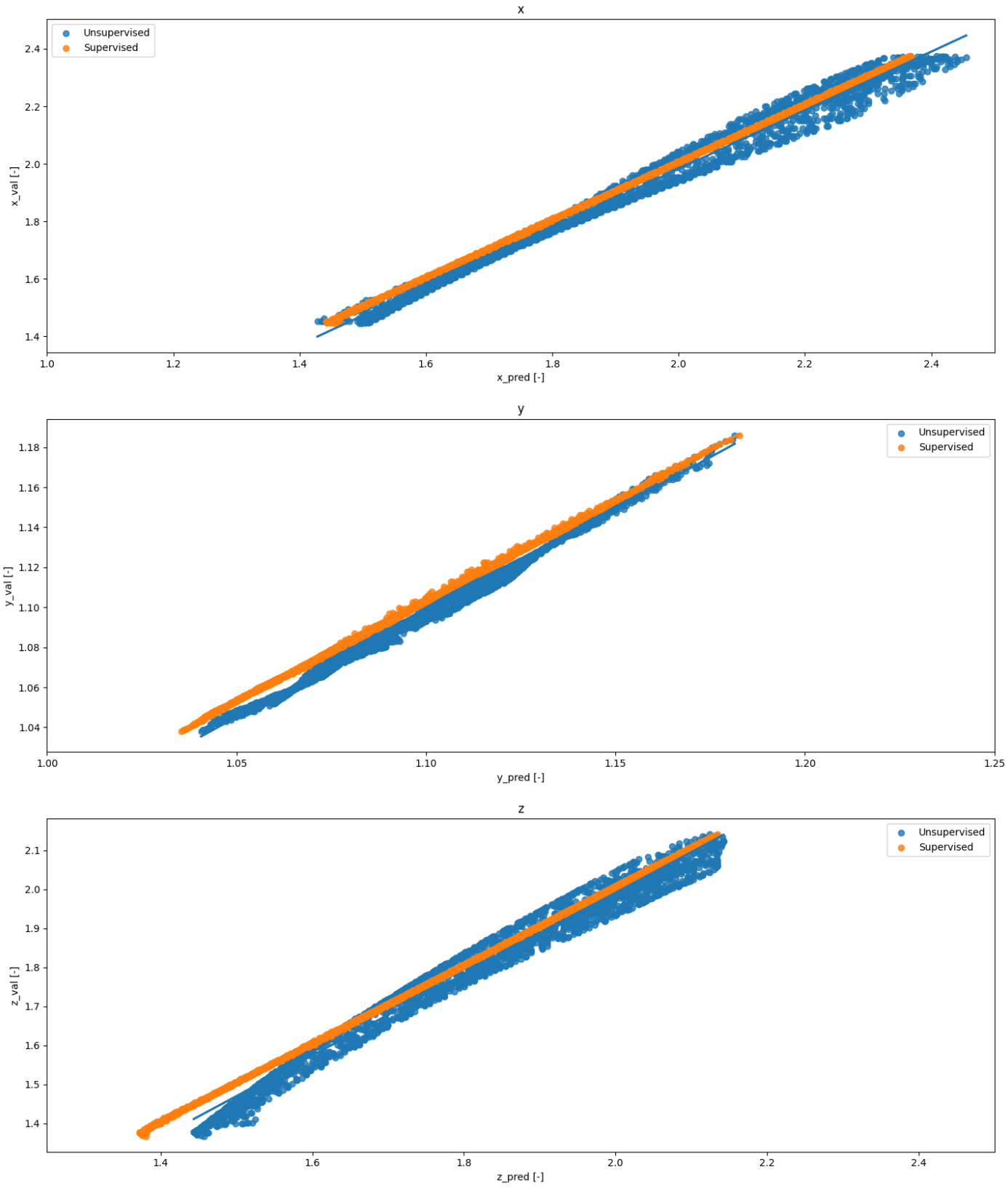

Figure A3: comparison of supervised and unsupervised approach. In each graph on the y-axis there are the predicted x,y,z-values of the NN and on the x-axis there are their corresponding validation x,y,z-values. The blue points refer to the predictions of the unsupervised NN and the orange points refer to the predictions of the supervised NN. The graph at the top refers to x-values, the graph in the middle refers to y-values and the graph at the bottom refers to z-values.

error	neurons per layer							
Relu-layers	3	50	100	200	400	500	700	900
1	0.0083382	0.4893976	0.001829	0.7008999	0.0405565	6.508393	2.6146193	0.0226663
2	0.4664292	0.0009788	0.0009968	0.6005204	0.0336701	0.0314854	0.0526878	0.0028329
3	0.0627234	0.0052211	0.0017702	0.0020415	0.0026134	0.0031992	0.0041353	0.0038599
4	0.0901263	0.0006361	0.0022501	0.0027775	0.000896	0.0026321	0.0018687	0.0200598

Table A1: Grid search with number of neurons per layer over Relu-layers

activation function	error
relu	0.00054485
tanh	0.14409877
sigmoid	0.00127387
exponential	nan
elu	2.8894193

Table A2: MSE for different activation functions

Hyperparameter						
epochs	norm	error	norm	error	no-norm	error
10	l1	0.0009497	l2	0.8289099		0.1294539
30	l1	0.0004683	l2	0.4189454		0.0003903
50	l1	0.0003574	l2	0.0004311	/	0.0022088
70	l1	0.000458	l2	0.0022246		0.0003985
100	l1	0.0018962	l2	0.0004761		0.000552
150	l1	0.0003511	l2	0.0004182		0.0002981

Table A3: l1,l2 and no-norm over different epochs

penalty_function_weights		
1 constraint	2 constraint	error
1	1	0.0020244
10	10	0.0123313
10	100	0.8489562
10	1000	1.8408579
100	10	0.0110287
100	100	1.4883475
100	1000	0.5835656
1000	10	0.0704627
1000	100	0.0246801
1000	1000	0.0183221
power2	power2	0.0038688
power3	power3	0.0006434
power4	power4	0.0010775
power5	power5	0.001573

Table A4: different weighings for constraints