CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# Efficient learning with privileged information in nonlinear time series

Master's thesis in computer science and engineering

Bastian Jung

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

# Efficient learning with privileged information in nonlinear time series

Bastian Jung

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Efficient learning with privileged information in nonlinear time series
Bastian Jung



Supervisor: Fredrik Johansson, Computer Science and Engineering
Examiner: Marina Axelson-Fisk, Mathematical Sciences

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2022

Efficient learning with privileged information in nonlinear time series
Bastian Jung
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

In domains where sample sizes are limited, efficient learning algorithms are critical. Learning using privileged information (LuPI) offers increased sample efficiency by allowing prediction models access to information at training time that is unavailable when the models are used. In recent work, it was shown that for prediction in linear-Gaussian dynamical systems, a LuPI learner with access to intermediate time series data is never worse and often better in expectation than any unbiased classical learner. We provide new insights into this analysis and generalize it to nonlinear prediction tasks in latent dynamical systems, extending theoretical guarantees to the case where the map connecting latent variables and observations is known up to a linear transform. In addition, we propose algorithms based on random features and representation learning for the case when this map is unknown. A suite of empirical results confirm theoretical findings and show the potential of using privileged time-series information in nonlinear prediction.

Keywords: Machine Learning, Privileged Information, Time Series, Sample Efficiency, Latent Dynamical Systems

# Acknowledgements

First, I need to express my gratitude towards my family and my beloved partner Pauline for their continued support during the challenging times that were the last two years. In addition, I need to thank my supervisor Fredrik, who has been an incredible source of inspiration and motivation to me. All of you have made this a fun journey, full of insights and moments that will certainly stay with me.

Bastian Jung, Berlin, May 2022

# Contents

# List of Figures

# List of Tables

# 1
# Introduction

Due to the vast amounts of data available in various fields, supervised machine learning has become a part of our daily lives. However, this data abundance is limited to certain domains. In healthcare, for example, machine learning practitioners have to deal with the problems of small data sets and high dimensionality [1]. Consequently, it is critical in these domains to make optimal use of all data available when creating prediction models, meaning there is a demand for sample efficient machine learning algorithms. When predicting the disease status of a patient at a set follow up time based on the data of a first medical examination, classical supervised learning techniques often use *only* the initial data for learning, even when informative data is generated at intermediate time steps. This could be information that is routinely collected after the first examination and before the follow up, such as further clinical tests or a patient's medication or vital signs. This data is often ignored in traditional supervised learning as it is unavailable when the model is finally used. The added data of intermediate time points is *privileged* as it can only be used for learning but not for making predictions [2]. Privileged information (PI) that occurs as intermediate steps of a time series, just like in the example, holds the potential of learning more accurate prediction models that require less data. We demonstrate sample efficient learning algorithms that leverage this additional information while predicting the outcome as a nonlinear function of the baseline input. Comparing to the alternative of classical learning, which ignores privileged time-series information, we identify conditions and algorithms for which privileged learning is clearly preferable.

**Context.** Improving the sample efficiency of prediction algorithms with side information has been approached from a variety of directions. Learning using privileged information (LuPI) was first presented as a machine learning paradigm by Vapnik & Vashisht [2] and is motivated by the analogy of a student learning with the help of a teacher. While the teacher can provide tips and tricks during the learning phase, the student must perform the intended task without external help eventually. Generalized distillation [3] has unified this approach with distillation by Hinton et al. [4], which can be seen as model compression technique that is also aimed at making more accurate predictions using a powerful teacher model. Multi-view learning [5] suggests learning to predict the same outcome from different perspectives to increase sample efficiency.

Existing theoretical results for learning from side-information guarantee improved learning rates [6][7] or give tighter generalization bounds [8] for large sample sizes

under appropriate assumptions. However, privileged information is not always beneficial: It must be related to the task of interest [9], otherwise the learning may be hindered rather than improved. However, previous work fails to identify for which exact settings learning with privileged information is provably preferable to classical learning. Moreover, previous works are uninformative about whether learning with PI has benefits in practical problems with small sample sizes, which is where efficiency is needed the most.

Karlsson et al. [10] studied LuPI in the context of predicting an outcome observed at the end of a time series based on variables collected at the first time step. They showed that making use of data from intermediate time steps in particular settings always leads to lower or equal prediction risk—for any sample size—compared to the best unbiased model which does not make use of this privileged information. However, their method called *learning using privileged time series* (LuPTS) was limited to settings where the outcome function of the data generating process, and estimators of it, are linear functions of baseline features observed at the start of the time series. Moreover, their analysis did not study how the variance reduction that is obtained from using privileged information behaves as a function of increased input dimension. Hayashi et al. [11] also learned from privileged intermediate time points but their study was limited to empirical results for classification using generalized distillation.

**Problem statement.** In the course of this project we aim to identify problem settings and algorithms where privileged time-series information is provably useful, meaning one can expect lower prediction error using a privileged learning algorithm compared to a classical learner that does *not* make use of privileged information. In particular we are interested in time series of the form $X_1, X_2, \ldots X_T, Y$, where intermediate steps $X_2, \ldots, X_T$ are considered privileged information and the task is to predict the outcome $Y$ as a nonlinear function of the baseline features $X_1$.

**Contributions.** This project extends the LuPTS framework of Karlsson et al. [10] to nonlinear models and prediction tasks in latent dynamical systems (Chapter 3). In this setting, we prove that learning with privileged information leads to lower risk compared to learning without it when the nonlinear map connecting latent variables and observations is known up to a linear transform. In doing so, we also find that when the representation dimension grows larger than the number of samples, the benefit of privileged information vanishes. We show that privileged learners used with random feature maps can learn optimal models consistently, even when the relationship between latent and observed variables is unknown, but may suffer from bias in small samples. As a remedy, we propose several representation learning algorithms aimed at trading off bias and variance. In experiments (Chapter 4) we find that privileged time-series learners with either random features or representation learning reduce variance and improve latent state recovery in small-sample settings on both synthetic and real-world regression tasks. In Chapter 5 the contributions of this project are discussed in the context of related work, while also pointing out promising directions for future research.

**Limitations.** While this project compares privileged against classical learning, the algorithms proposed and analyzed are only compared to alternatives within the same hypothesis class. This means we do not compare the predictive performance of our algorithms against current state-of-the-art methods for long-term time series prediction. It is not the goal to propose a single best model for the regression tasks considered. Instead the project intends to demonstrate the potential of incorporating privileged information into the training processes of prediction models that are often used in practice today and that do not make use of this additional information yet. Further, we do not analyze the benefit of privileged information in general, the project scope is restricted to discrete time series of a fixed length where the outcome is observed at the end of such a sequence. While the theoretical results presented make assumptions about the data generating process we discuss the strength of these assumptions and also test resulting algorithms on real-world data where the conditions demanded by our assumptions cannot be guaranteed.

**Notation.** We denote random variables or random vectors using capitalized letters $X$. A value drawn from such a random variable is given in lowercase letters $x$. Design matrices containing the samples of random variable $X$ are denoted by bold type $\mathbf{X}$. Spaces that samples $x$ reside in are presented using calligraphic characters $\mathcal{X}$.

# 2
# Background

## 2.1 Learning using privileged information

Learning using privileged information was introduced as a paradigm of machine learning in 2009 by Vapnik & Vashisht [2] and has been used in many different ways for a variety of applications. It describes an extension of the popular machine learning paradigm of supervised learning. The inventors of this approach motivated their work with the analogy of a student learning with a helping teacher. During the learning process the teacher will provide additional information to the tasks to be solved, such as explanations, comments and hints (privileged information). For instance, students might be told which examples are similar. The student will consider this side information in order to facilitate the learning. In the end however, the student is expected to solve the same class of tasks without the added support given by the teacher.

Supervised learning is concerned with learning a model $\hat{f} \in \mathcal{F} : \mathcal{X} \to \mathcal{Y}$, predicting an outcome $Y \in \mathcal{Y}$ from a variable $X \in \mathcal{X}$. It is the goal to find a function $\hat{f}$ that is in some sense close to the true function $f$ that describes the relationship of $Y$ and $X$ which have joint distribution $p(X, Y)$. Note that $f$ might not be deterministic. To find a function $\hat{f}$ the goal would often be to minimize the generalization error

$$R(\hat{f}) = \mathbb{E}_{X,Y}\left[\mathcal{L}(Y, \hat{f}(X)\right] , \tag{2.1}$$

where $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a real-valued loss function describing the similarity of the two functions when applied to the random variable $X$. In a typical machine learning setting, one does not have access to the true distribution $p(X, Y)$ which makes direct minimization of $R(\hat{f})$ impossible. Instead one can minimize the empirical risk $\hat{R}(\hat{f})$. Let $(x_i, y_i)$ denote a sample of $p(X, Y)$, where $X$ is the independent variable while $Y$ is the dependent variable of a regression problem. Assuming we observe $m$ samples of the random variables $X$ and $Y$, the empirical risk can be written as

$$\hat{R}(\hat{f}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{f}(x_i), y_i) \ . \tag{2.2}$$

Consequently, a learning algorithm can minimize the empirical risk as a proxy for the generalization error by finding a function

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \hat{R}(\hat{f}) \ . \tag{2.3}$$

While the supervised machine learning approach consists of learning from samples of the form $(X, Y)$, learning from privileged information means learning from triplets $(X, Z_P, Y)$, where $Z_P \in \mathcal{Z}_P$ is privileged information which means it is only observed during training. For both paradigms the goal is to learn $\mathbb{E}[Y|X]$ but in the PI case one hopes that the privileged information $Z_P$ is informative about the conditional distribution $p(Y|X)$ and thus allows for a better estimate of this expectation for a given amount of data.

To demonstrate the idea behind learning from privileged information consider the following example: Imagine a smart measuring device recording mechanical vibrations while mounted on an aircraft turbine. The task is to detect dangerous turbine states in advance to warn pilots about a potential equipment failure ahead of time. Let dangerous states be denoted with $Y = 1$ while $Y = 0$ are normal turbine conditions. When making predictions the device can only access the signals of the sensors it is equipped with, which are the vibration measurements $X$. During the development of this device the engineers developing the turbine may use a large array of different and expensive sensors to measure sound, temperature, air flow and other variables in addition to the signals coming from the smart measuring device. These additional variables are not available when the prediction model in the smart measurement device will be used, they are *privileged information* $Z_P$ that can only be incorporated into the learning process. The hope is that the additional data will improve the learning, such that the smart device makes more accurate predictions when it is used as a finished product despite not having access to the same variables as during training when the turbine was developed.

When this new paradigm of machine learning was introduced, Vapnik et al. presented a modified version of support vector machines (SVM) that benefits from privileged information. In particular the added information was used to predict the slack variables corresponding to each training example. The modified version was found to deliver more accurate results on classification tasks compared to the classical SVM alternative and therefore first demonstrated the potential of learning from privileged information.

## 2.2 Distillation

Learning from privileged information is closely related to the concept of distillation that was introduced by Hinton et al. [4]. The problem addressed by this technique was that very often a large cumbersome model would perform better on some given task while a simpler model is more computationally efficient and might be easier to implement on limited hardware. The idea behind distillation is to first produce so called *soft targets* by training a large model on the training data. In a second step one then trains the smaller model using a modified loss function which compares the predictions to the labels of the training data as usual but also to the soft targets.

Consider predicting the binary labels $Y$ from $X$ in a classification task. Neural networks usually produce a logit $f(X_i)$ as an output that can be converted into a

class probability $q_i$ via the softmax function or the logistic function $\sigma(x)$ for a binary classification setting. Let $f$ be a classifier such that $q_i = \sigma(f(X_i))$ is the probability of a given example belonging to class one. An optimal classifier minimizes the cross-entropy loss

$$\mathcal{L}(Y, \sigma(X)) := -Y \log(\sigma(f(X)) - (1 - Y) \log(1 - \sigma(f(X))). \tag{2.4}$$

In distillation one minimizes the empirical cross entropy loss using a class of very flexible models $\mathcal{F}_T$ to find a well performing model $\hat{f}_T$. The index $T$ denotes that this flexible model has the role of a teacher.

$$\mathcal{L}(Y_i, \sigma(f(X_i))) = -Y_i \log(\sigma(f(X_i)) - (1 - Y_i) \log(1 - \sigma(f(X_i))) \tag{2.5}$$

$$\hat{f}_T = \arg\min_{f \in \mathcal{F}_T} \sum_{i=1}^{n} \mathcal{L}(Y_i, \sigma(f(X_i))) \tag{2.6}$$

In the second step one would then use the class probabilities produced by the flexible model $s_i := \sigma(\hat{f}_a(X_i)))$ as an additional soft target to train a model that belongs to a more restrictive class of functions $\mathcal{F}_S$. This model is also called a student model and minimizes the combined loss

$$\hat{f}_S = \arg\min_{f \in \mathcal{F}_S} \sum_{i=1}^{n} (1 - \lambda) \mathcal{L}\Big(Y_i, \sigma(f(X_i)\Big) + \lambda \mathcal{L}\Big(s_i, \sigma(f(X_i))\Big) \ . \tag{2.7}$$

The hyperparameter $\lambda \in (0, 1)$ is used to trade off the importance of fitting the training data versus matching the soft targets. The intuition behind this approach is that a lot of useful information is contained in the soft targets which is difficult to extract from the labels.

Lopez-Paz et al. later unified the concept of distillation with learning from privileged information [3], calling the new framework generalized distillation. In essence, the authors argue that both approaches first learn a teacher function $f_T$ on samples of the form $p(Z, Y)$ to produce some form of soft targets before fitting a student function $f_S$. In the case of Vapnik's approach, $Z$ is privileged information and does not match the explanatory variable $X$. In contrast Hinton's distillation sets $Z = X$ and trains the teacher function on samples from $p(X, Y)$. In generalized distillation the student function $f_S$ is then trained on a combination of the original targets $Y_i$ and the soft targets $s_i$ provided by the teacher function, while balancing the two objectives with the imitation parameter $\lambda$ just as shown in objective 2.7.

The motivation behind these two approaches presented in this section differ despite their common structure. In distillation one uses a very rich teacher model to extract a very useful representation from large amounts of baseline data $X$. Such a representation would be hard to extract with a simple model. Vapnik's approach does not require a rich teacher model but instead aims to make use of the very rich representation offered by the privileged information $Z$ that is either hard to extract or not present in the baseline data $X$. This is done to use a given amount of data more efficiently.

## 2.3 Ordinary least squares regression

The ordinary least squares estimator finds a parameter $\theta$ that minimizes the empirical risk given by the mean squared error $\mathcal{L}$ over a given data set consisting of $m$ tuples of the form $(X, Y)$ where $X \in \mathcal{X} \subseteq \mathbb{R}^k$ and $Y \in \mathcal{Y} \subseteq \mathbb{R}^q$. The ordinary least squares estimator is the maximum likelihood estimator for the case where $Y$ is a linear function of $X$ with added Gaussian noise with zero mean and covariance $\sigma^2 I$, where $I$ is the $q \times q$ identity matrix.

$$Y := \theta^\top Y + \epsilon \tag{2.8}$$

$$\epsilon \sim N(0, \sigma^2 I) \tag{2.9}$$

The likelihood $L$ of observing the data set $D$ consisting of independent identically distributed samples $(X_i, Y_i)$ drawn from the distribution $p(X, Y)$ is

$$L = \prod_{i=1}^{m} p(X_i, Y_i) \tag{2.10}$$

$$= \prod_{i=1}^{m} p(Y_i|X_i)p(X_i) \tag{2.11}$$

$$= \prod_{i=1}^{m} \det(2\pi\sigma I)^{-\frac{1}{2}} \exp\Big( -\frac{1}{2}((Y_i - \theta^\top X_i)^\top \sigma I (Y_i - \theta^\top X_i))\Big) p(X_i) \tag{2.12}$$

To obtain the maximum likelihood estimator, one can minimize the negative log likelihood.

$$-\log(L) \propto \sum_{i=1}^{m} \frac{1}{2\sigma^2}(Y_i - \theta^\top X_i)^\top (Y_i - \theta^\top X_i) + \log(p(X_i)) \tag{2.13}$$

$$= \frac{1}{2\sigma^2} \sum_{i=1}^{m} (Y_i - \theta^\top X_i)^\top (Y_i - \theta^\top X_i) + \sum_{i=1}^{n} \log(p(X_i)) \tag{2.14}$$

$$= \frac{1}{2\sigma^2} \sum_{i=1}^{m} \|(Y_i - \theta^\top X_i)\|_2^2 + \sum_{i=1}^{m} \log(p(X_i)) \tag{2.15}$$

Consequently, the maximum likelihood estimator minimizes the sum of squared residuals $\sum_{i=1}^{m} \|(Y_i - \theta^\top X_i)\|_2^2$. In the following the OLS estimator is derived. Let $\mathbf{X} \in \mathbb{R}^{m \times k}$ and $\mathbf{Y} \in \mathbb{R}^m$ be a data set with $m$ samples and $k$ features for $X$ and a single feature for $Y$ without loss of generality.

$$\min_{\theta} \quad \mathcal{L} := \frac{1}{2} \| \mathbf{Y} - \mathbf{X}\,\theta \|_2^2 \tag{2.16}$$

Taking the derivative with respect to $\theta$ and setting it to zero one gets the following optimality condition for this convex optimization problem:

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\,\theta) = 0 \tag{2.17}$$

$$\iff \mathbf{X}^\top \mathbf{X}\,\theta = \mathbf{X}^\top \mathbf{Y} \tag{2.18}$$

If one has linearly independent features and $k \leq m$, then $\mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{k \times k}$ must be invertible. In this case the unique ordinary least squares (OLS) solution is:

$$\theta_{OLS} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y} \tag{2.19}$$

However, it might be the case that one has more features than samples $m < k$. In this case one can show that a solution to the condition in equation 2.18 is

$$\theta_{OLS+} = (\mathbf{X}^\top \mathbf{X})^{\dagger} \mathbf{X}^\top \mathbf{Y} \ , \tag{2.20}$$

where $(\cdot)^\dagger$ is the Moore-Penrose pseudoinverse that replaces the regular matrix inverse $(\cdot)^{-1}$ [12]. The solution $\theta_{OLS+}$ for the underdetermined case is not unique. In other words, there are many hyperplanes that fit the training data perfectly. To see this, notice that it was assumed that $k > m$ which means that $\mathbf{X}$ cannot have full column rank. This is equivalent to saying the nullspace of $\mathbf{X}$ is not empty. Now one can see that any vector $v \in \mathbb{R}^k$ that lies in the nullspace of $\mathbf{X}$ can be added to $\theta_{OLS+}$ and this new solution still satisfies the optimality condition of equation 2.18:

$$\mathbf{X}^\top \mathbf{X}(\theta_{OLS+} + v) = \mathbf{X}^\top \mathbf{Y} \tag{2.21}$$

$$\iff \mathbf{X}^\top \mathbf{X}\, \theta_{OLS+} + \mathbf{X}^\top \underbrace{\mathbf{X}\, v}_{=0} = \mathbf{X}^\top \mathbf{Y} \tag{2.22}$$

$$\iff \mathbf{X}^\top \mathbf{X}\, \theta_{OLS+} = \mathbf{X}^\top \mathbf{Y} \tag{2.23}$$

However, it can be shown that there is always a solution that has the smallest norm $\|\theta\|_2^2$ and that this solution is exactly $\theta_{OLS+}$. Notice that the solutions $\theta_{OLS}$ and $\theta_{OLS+}$ are almost the same. In fact, these solutions can be given in one unified expression:

$$\theta_{OLS} = (\mathbf{X}^\top \mathbf{X})^{\dagger} \mathbf{X}^\top \mathbf{Y} \tag{2.24}$$

When we assume $k \leq m$, then we know $\mathbf{X}^\top \mathbf{X}$ is invertible. In this case the Moore-Penrose inverse is identical to the regular matrix inverse, i.e. $(X^\top X)^\dagger = (X^\top X)^{-1}$, and only one unique solution exists. When $k > m$ there are many possible solutions and equation 2.24 gives the minimum norm solution of the least squares problem.

## 2.4 Bias-variance decomposition

When using empirical risk minimization to fit a hypothesis $\hat{h} : \mathcal{X} \subseteq \mathbb{R}^k \to \mathcal{Y} \subseteq \mathbb{R}$, the expected risk with squared error loss $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ can be decomposed into three parts that provide a principled framework to discuss the characteristics of prediction models: Bias, variance and irreducible noise [13]. We use the squared error loss

$$\mathcal{L}(y, \hat{h}_D(x)) \coloneqq (y - \hat{h}(x))^2 \tag{2.25}$$

and consider the prediction risk, that is the expected loss

$$\mathbb{E}_{D,X,Y}\Big[\mathcal{L}(Y,\hat{h}_D(X))\Big] = \mathbb{E}_D\Bigg[\int \mathcal{L}(h(x),\hat{h}_D(x))p(x)dx + \underbrace{\int\int \mathcal{L}(h(x),y)p(x,y)dxdy}_{\text{noise}}\Bigg] . \tag{2.26}$$

The true hypothesis one wants to learn is $h(\cdot)$, while it is corrupted by noise such that we observe $y_i$ instead of $h(x_i)$ in our data sets $D \in \mathcal{D}$ with $D \coloneqq \{(x_i, y_i)\}_{i=1}^m$. The second term of (2.26) is irreducible noise in the labels and does not depend on the predictions $\hat{h}(x)$. It is the same for all prediction models. We turn to the first term and derive the bias-variance decomposition for a fixed point $x \in \mathcal{X}$ where $\hat{h}_D(x)$ is our model evaluated at $x$ after being trained on a randomly chosen fixed data set $D$.

We may expand the first term of (2.26) by adding and subtracting $\mathbb{E}_D[\hat{h}_D(x)]$, obtaining

$$\begin{aligned}\mathcal{L}(h(x),\hat{h}_D(x)) =& \Big(h(x) - \mathbb{E}_D[\hat{h}_D(x)] + \mathbb{E}_D[\hat{h}_D(x)] - \hat{h}_D(x)\Big)^2 \\ =& \Big(\hat{h}_D(x) - \mathbb{E}_D[\hat{h}_D(x)]\Big)^2 + \Big(\mathbb{E}_D[\hat{h}_D(x)] - h(x)\Big)^2 \\ &+ 2\Big(\hat{h}_D(x) - \mathbb{E}_D[\hat{h}_D(x)]\Big)\Big(\mathbb{E}_D[\hat{h}_D(x)] - h(x)\Big) .\end{aligned} \tag{2.27}$$

Now one can compute the expectation over data sets $D \in \mathcal{D}$ which lets the final term vanish.

$$\mathbb{E}_D(\mathcal{L}(h(x),\hat{h}_D(x))) = \underbrace{\Big(\mathbb{E}_D\Big[\hat{h}_D(x) - h(x)\Big]\Big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_D\Big[\Big(\hat{h}_D(x) - \mathbb{E}_D[\hat{h}_D(x)]\Big)^2\Big]}_{\text{variance}} \tag{2.28}$$

Substituting this back into (2.26) one gets

$$\text{expected loss} = \text{bias}^2 + \text{variance} + \text{noise} ,$$

which is what is commonly referred to as the bias-variance decomposition. Bias describes the difference between the mean prediction made by a model for a given point $x$ and the true function at that point. Variance describes the deviation of predictions around the mean prediction of the model. In the course of this project we analyze learning algorithms in terms of the bias and variance of their hypotheses in expectation over data sets.

## 2.5 Kernel methods

Linear methods have many advantages. They are easy to implement and there exists a closed form optimal solution in the case of ordinary least squares, ridge regression [14] and others. Further, one can interpret a trained model in a straightforward way

by analyzing the coefficients of $\hat{\theta}$. However, linear models often lack the expressiveness that is required to solve certain problems. Often the true function one is trying to learn is just not linear. As a result, there is a need to extend linear methods to make non-linear predictions with similar algorithms.

**Feature maps.** One possible extension of linear methods is the use of feature maps. Imagine trying to learn the following function.

$$f : \mathbb{R} \to \mathbb{R} \tag{2.29}$$
$$Y = f(X) = \theta_0 X^2 + \theta_1 X + \theta_2 \tag{2.30}$$

If one only has samples consisting of tuples of the form $(x, y)$ a linear model $\hat{h}(x) = \theta^\top x$ does not fit this data very well as $Y$ is not linear in $x$. One easy solution is to use a feature map $\Phi : \mathcal{X} \to \mathcal{H}$ in order to project the data into a different space. For this example one could use the map

$$\Phi(x) = \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix}. \tag{2.31}$$

Applying the linear estimation method on the modified data, which are samples of $(\Phi(x), y)$, can yield a much lower generalization error as the estimator $\hat{h}(\cdot) = \theta^\top \Phi(\cdot)$ is now in the same hypothesis class $\mathcal{H}$ as the true function $f$, meaning the estimator now has the capability to represent the true function which was not the case before using the feature map. We call this the realizable case and refer to the non-realizable case when the true function $f$ is not in the hypothesis class $\mathcal{H}$ of the estimator $\hat{h} \in \mathcal{H}$.

Although it is useful in making predictions that are a nonlinear in the input, the approach of using feature maps poses new problems. For instance it is not obvious how to choose the basis functions that make up the feature map $\Phi$ in order to produce features that facilitate the learning. This might also be highly problem specific. For this reason one might want to include very many basis functions with leads to very high dimensional feature spaces which in turn might make the computation of the feature map expensive.

**The kernel trick.** Coming from the perspective of feature maps one might want to avoid the explicit computation of the features $\Phi(x)$. Many algorithms can be expressed in terms of dot products between training samples only. Consequently, it would be very useful to have access to a function $\kappa :=: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which computes the dot product $\langle \cdot\ ,\ \cdot \rangle_{\mathcal{H}}$ between two samples $x$ and $x'$ in the feature space $\mathcal{H}$ directly:

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \tag{2.32}$$
$$\kappa(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}} \tag{2.33}$$

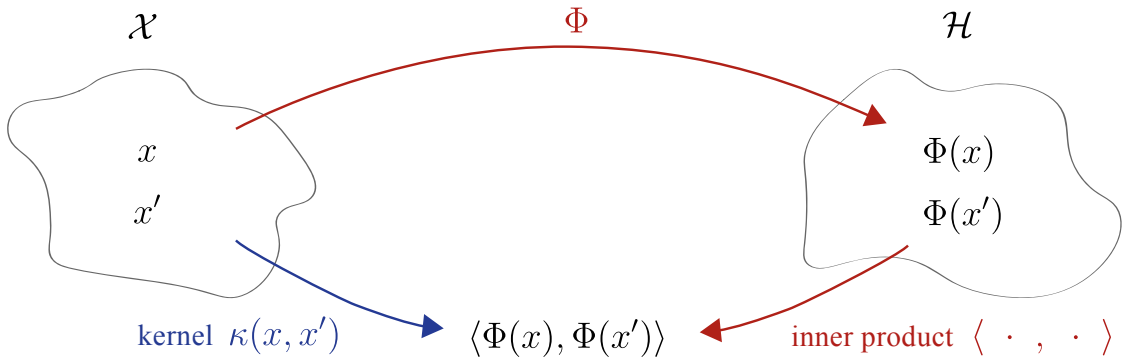



**Figure 2.1:** Feature space $\mathcal{X}$ and the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ of a reproducing kernel $\kappa$. Instead of first projecting to the RKHS using a the feature map $\Phi$ before computing an inner product, one may make use of the kernel function $\kappa$ directly. This is especially useful if the feature space of the RKHS is high-dimensional or not even finite.

Such a kernel function $\kappa$ therefore offers a shortcut if the estimation algorithm used requires only the computation of dot products between samples. Rather than having to apply the feature map in order to get $\Phi(x)$ and $\Phi(x')$ before then computing the inner product $\langle \cdot , \cdot \rangle_{\mathcal{H}}$, one can simply compute $\kappa(x, x')$ instead. This notion is illustrated by Figure 2.1.

The idea behind what is often referred to as the kernel trick is to find a way to express an estimator only in terms of dot products between samples in $\mathcal{X}$ and then to replace those dot products by the kernel function. As a consequence one does not even need to know the explicit feature map $\Phi$ in order to use a linear prediction method on feature vectors that lie in the space $\mathcal{H}$.

**Kernel functions as similarity measures in linear spaces.** Scalar products can be regarded as a similarity measure between data points. The same intuition can be applied to kernel functions which can be viewed as generalized dot products. This means that $\mathcal{X}$ can be any space and is not restricted to $\mathbb{R}^d$. One can therefore apply kernel machines on data of any kind as long as one has access to a similarity measure between different examples. After having presented the broad ideas behind using kernel functions, their properties and definitions shall be shown in more detail. Most definitions and concepts presented in this subsection are restated from the work of Schölkopf and Smola [15].

**Kernel matrix.** At first we define the kernel matrix $\mathbf{K}$ which is also sometimes called the Gram matrix. Given a function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and data points $x_1, x_2, ..., x_n \in \mathcal{X}$, it is defined as the $m \times m$ matrix with elements

$$\mathbf{K}_{ij} := \kappa(x_i, x_j) \ . \tag{2.34}$$

**Positive definite kernels.** A kernel function $\kappa$ is called positive definite if it gives rise to a positive-semi-definite kernel matrix $\mathbf{K}$. When a kernel produces a positive-definite kernel matrix, meaning the matrix has only positive eigenvalues, it shall be called strictly positive definite. From this definition we get the following properties of positive definite kernels.

The diagonal elements of the kernel matrix $\mathbf{K}$ must be non-negative, meaning

$$\kappa(x_i, x_i) \geq 0 \tag{2.35}$$

and the kernel must be symmetric:

$$\kappa(x_i, x_j) = \kappa(x_j, x_i) \tag{2.36}$$

**Reproducing kernel Hilbert spaces.** A reproducing kernel Hilbert space (RKHS) is a space of functions that is a realization of the feature space associated with a particular kernel function $\kappa$. Instead of thinking about the feature map $\Phi$ applied to every data point $x_i$ one can picture the RKHS as a space in which each pattern $x_i$ is represented by a function $g : \mathcal{X} \to \mathbb{R}$ that is parametrized by $x_i$, such that $g(\cdot) = \kappa(x_i, \cdot)$. As with all linear spaces $\mathcal{H}$ contains linear combinations of those functions:

$$f = \sum_i \alpha_i \kappa(x_i, \cdot) \tag{2.37}$$

$$\text{with } f \in \mathcal{H}, x_i \in \mathcal{X}, \alpha \in \mathbb{R} \tag{2.38}$$

Here the sum uses the index $i$ to point out the correspondence of the scalar $\alpha_i$ and a particular $x_i \in \mathcal{X}$. However, the sum can be understood as a sum over all $x \in \mathcal{X}$. One can give the following formal definition for the RKHS:

**Definition 1** (Reproducing Kernel Hilbert Space)**.** *Let $\mathcal{X}$ be a nonempty set and $\mathcal{H}$ a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called a reproducing kernel Hilbert space which has a dot product $\langle \cdot, \cdot \rangle$ and the norm $||f|| = \sqrt{\langle f, f \rangle}$ if there exists a kernel function $\kappa$ with the following properties:*

- *$\kappa$ has the reproducing property: $\langle f, \kappa(x, \cdot) \rangle = f(x) \forall f \in \mathcal{H}$*
  *which also means $\langle \kappa(x_i, \cdot), \kappa(x_j, \cdot) \rangle = \kappa(x_i, x_j)$*
- *$\kappa$ spans $\mathcal{H}$ which means $\mathcal{H} = \overline{span\{\kappa(x, \cdot) | x \in \mathcal{X}\}}$ where $\overline{X}$ denotes the completion of a set $X$, meaning that all Cauchy sequences converge within $\mathcal{H}$*

**The reproducing property.** The reproducing property of positive definite kernels means that $\kappa$ is the representer of evaluation. Computing the dot product between some function $f \in \mathcal{H}$ and the function $\kappa(x, \cdot)$ is the same as evaluation of $f$ at $x$.

$$\langle f, \kappa(x, \cdot) \rangle = f(x)$$

The proof is simple. Let $f = \sum_i \alpha\kappa(x_i, \cdot)$ be a functional in the RKHS $\mathcal{H}$.

$$\langle f, \kappa(y, \cdot)\rangle = \langle \sum_i \alpha\kappa(x_i, \cdot), \kappa(y, \cdot)\rangle \tag{2.39}$$

$$= \sum_i \alpha\langle\kappa(x_i, \cdot), \kappa(y, \cdot)\rangle \tag{2.40}$$

$$= \sum_i \alpha\kappa(x_i, y)\rangle \tag{2.41}$$

$$= f(y) \tag{2.42}$$

The second equality uses the linearity of the inner product while the third uses the fact that the kernel function is defined to resemble dot products between elements in the RKHS.

**The representer theorem.** Although the RKHS $\mathcal{H}$ might have infinite dimensions, the representer theorem tells us that for regularized convex optimization problems like described below, one can find a solution that can be expressed as a kernel expansion of the training data [16]. This means that the solutions of empirical risk minimization problems in the high dimensional RKHS always lie in the span of the training data.

**Theorem .** Let $\mathcal{X}$ be a non-empty set, let $k$ be a positive-definite kernel on $\mathcal{H} \times \mathcal{H}$ and let $(x_1, y_1), (x_2, y_2), ...(x_n, y_n) \in \mathcal{H} \times \mathbb{R}$. Further, let $g$ be a strictly monotonically increasing real-valued function on $[0, \infty]$ and consider an arbitrary cost function $c : \mathcal{H} \times \mathbb{R}^2 \to \mathbb{R} \cup \{\infty\}$ and a class of functions $\mathcal{F}$

$$\mathcal{F} = \left\{ f \in \mathbb{R}^{\mathcal{X}} \;\middle|\; f(\cdot) = \sum_{i=1}^{\infty} \beta_i \kappa(z_i, \cdot),\ \beta_i \in \mathbb{R}\,,\ z_i \in \mathcal{X},\ \|f\| < \infty \right\} \tag{2.43}$$

Here $\mathbb{R}^{\mathcal{X}}$ is the space of functions from $\mathcal{X}$ to $\mathbb{R}$. The norm $\|\cdot\|$ is the one associated with the RKHS $\mathcal{H}$ and the kernel $\kappa$, such that for any $z_i \in \mathcal{X},\ \beta_i \in \mathbb{R},\ i \in \mathbb{N}$

$$\left\| \sum_{i=1}^{\infty} \beta_i \kappa(z_i, \cdot) \right\|^2 = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \beta_i \beta_j \kappa(z_i, z_j)\ . \tag{2.44}$$

Then any $f \in \mathcal{F}$ minimizing the regularized risk functional

$$c((x_1, y_1, f(x_1)), \ldots, (x_n, y_n, f(x_n))) + g(\|f\|) \tag{2.45}$$

admits a representation of the form

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot) \tag{2.46}$$

## 2.6 Kernelizing ordinary least squares

In order to use the OLS estimator in a reproducing kernel Hilbert space $\mathcal{H}$, we need to be able to express a new prediction $\hat{y} = \theta_{OLS}^{\top} x$ only in terms of dot products between

the training samples in $\mathcal{X}$ which are arranged into a design matrix $\mathbf{X} \in \mathbb{R}^{m \times k}$, containing $m$ samples and $k$ features. Further, we want the new test sample $x \in \mathbb{R}^k$ also to only occur in terms of dot products with training samples. This can be achieved by using the following matrix identity:

$$(\mathbf{A}^\top \mathbf{A})^\dagger \mathbf{A}^\top = \mathbf{A}^\top (\mathbf{A}\mathbf{A}^\top)^\dagger \tag{2.47}$$

This identity is proven simply by using singular value decomposition on $A$ and using some properties of the Moore-Penrose pseudoinverse. Applying it to the general OLS estimator given in equation 2.24 yields an expression where the matrix multiplication between training samples always takes place in the feature dimension:

$$\hat{h}_{OLS}(x) = \theta_{OLS}^\top x = ((\mathbf{X}^\top \mathbf{X})^\dagger \mathbf{X}^\top \mathbf{Y})^\top x \tag{2.48}$$
$$= (\mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^\dagger \mathbf{Y})^\top x \tag{2.49}$$
$$= \mathbf{Y}^\top (\mathbf{X}\mathbf{X}^\top)^\dagger \mathbf{X}\, x \tag{2.50}$$

Instead of fitting a linear function in $\mathbb{R}^k$, one can now do so in the space that one gets after applying the feature map $\Phi : \mathcal{X} \to \mathcal{H}$ to the data. This function shall correspond to a kernel $\kappa$ such that $\kappa(x, x') :== \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}$. $\mathbf{Z}$ refers to the $m \times d$ design matrix that one gets by applying the feature map $\Phi(\cdot)$, which corresponds to kernel $\kappa$, to each row of $\mathbf{X}$. $d$ is the resulting number of features and is the dimensionality of the RKHS. Note that the feature map may not always be known and that $d$ might not be finite for some kernel functions. Analogously, let $z$ denote this feature map applied to the test vector $x$. Instead of using the feature map explicitly, one can use the kernel trick which means replacing $\mathbf{Z}\mathbf{Z}^\top$ by $\mathbf{K}$ as specified in equation 2.34. Kernelized ordinary least squares therefore takes on the form:

$$\theta_{OLS} = \mathbf{Z}^\top (\mathbf{Z}\mathbf{Z}^\top)^\dagger \mathbf{Y} \tag{2.51}$$
$$\implies \theta_{OLS}^\top \mathbf{z} = \mathbf{Y}^\top (\mathbf{Z}\mathbf{Z}^\top)^\dagger \mathbf{Z} z \tag{2.52}$$
$$\boldsymbol{\alpha}_{OLS} := (\mathbf{Z}\mathbf{Z}^\top)^\dagger \mathbf{Y} = \mathbf{K}^\dagger \mathbf{Y} \tag{2.53}$$
$$\hat{h}_{OLS}(x) = \sum_{i=1}^{n} \boldsymbol{\alpha}_i \kappa(\mathbf{X}_i, x) \tag{2.54}$$

From this expression one can see that by kernelizing the OLS estimator one gets a non-parametric version of this estimator where the similarity between the training samples $\mathbf{X}_i$ and the new input $x$ are computed to make a new prediction. Note also that one does not require access to the feature map $\Phi$ as the estimator can be computed in terms of the kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ used on training examples in $\mathcal{X}$ only. However, one needs to have access to all training data when making a new prediction, meaning after training the data needs to be kept in memory. This is different for the parametric version, where one uses a feature map instead of a kernel and stores a parameter vector $\theta_{OLS}$ but not the data.

## 2.7 Deep representation learning

How information is presented can make information processing tasks more difficult or much easier. This general principle is the motivation for representation learning which is concerned with discovering a more useful representation of the data that makes subsequent learning tasks easier. Many problems can be hard or easy depending on how they are posed. As an example imagine the task of learning to separate points inside the unit circle in $\mathbb{R}^2$ from the ones outside. If the data is given in cartesian coordinates, this problem is clearly not linearly separable and the data representation seems unsuited for the task at hand. If we change the representation to polar coordinates, the task becomes linearly separable in one dimension, which is a huge simplification as this problem can now be solved with linear model. Just like the transformation to polar coordinates, good representations are ones that make subsequent learning tasks easier [17].

One can therefore view supervised learning on deep neural networks as representation learning: The last layer of a multi-layer perceptron can be viewed as solving a linear regression problem while the layers before have the task to find a representation of the data that makes a linear model feasible.

With supervised training one does not impose any restrictions on the learnt representation other than it being useful for the task at hand i.e. regression or classification. However, one often wants to do unsupervised representation learning in order to benefit from large amounts of unlabeled data that is more readily available compared to labelled data. In a second step one uses the learned representation to solve a supervised learning problem. In this context one wants the representation to have desirable properties which means imposing restrictions. A good example of this approach are variational autoencoders [18], which learn to represent the input data with independent latent variables. In this case the independence of these latent features is a useful restriction since it makes them much easier to analyze.

Representation learning with neural networks plays a key role in many different domains such as natural language processing [19], graph learning [20], computer vision [21] and others. As we make use of neural networks when combining representation learning with LuPI in Chapter 3.4, we also give a very brief introduction to learning with neural architectures as well.

### 2.7.1 Neural networks

The perceptron invented by Rosenblatt in 1958 is often seen as the starting point for machine learning with neural networks [22]. Simple feedforward neural networks are composed of layers of neurons. Each neuron in such a network is connected to every neuron in the previous layer and every neuron in the following layer of the network. During a forward pass through the network, which means computing the hypothesis $\hat{h}(x)$ for a given input $x$, the states of the neurons in the first layer are set to the values of the input, then the layers of neurons successively update their

states until the output is obtained from the neuron states in the last layer. Every neuron $i$ in layer $l$ is parameterized by a weight vector $\mathbf{W}_{(l,j)}$ and a bias scalar $b_{(i,l)}$. Each neuron computes its state $\boldsymbol{s}_{(l,i)}$ using the McCulloch-Pitts dynamics during a forward pass [23]:

$$\boldsymbol{s}_{(l,i)} = \sigma(\mathbf{W}_{(l,i)}^{\top}\boldsymbol{s}_{(l-1,:)} - b_{(l,i)}) \tag{2.55}$$

The non-linear function $\sigma$ is called an activation function. The name is due to the notion of certain neurons being active when their output is non-zero which is analogous to the activation of neurons in the mammalian brain. Therefore, the function $\sigma$ which is usually the same for all neurons determines for which inputs a neuron is activated and for which it remains inactive. Popular activation functions are the rectified linear unit (ReLU) and the sigmoid function. A review of different activation functions and their characteristics are found in [24].

It has been shown that multi-layer perceptrons (MLP) such as displayed in Figure 2.2 have universal function approximation capabilities which means that by tuning the weights $\mathbf{W}_l$ and bias terms $b_l$ an MLP can approximate any continuous function arbitrarily well given the network is large enough [25].

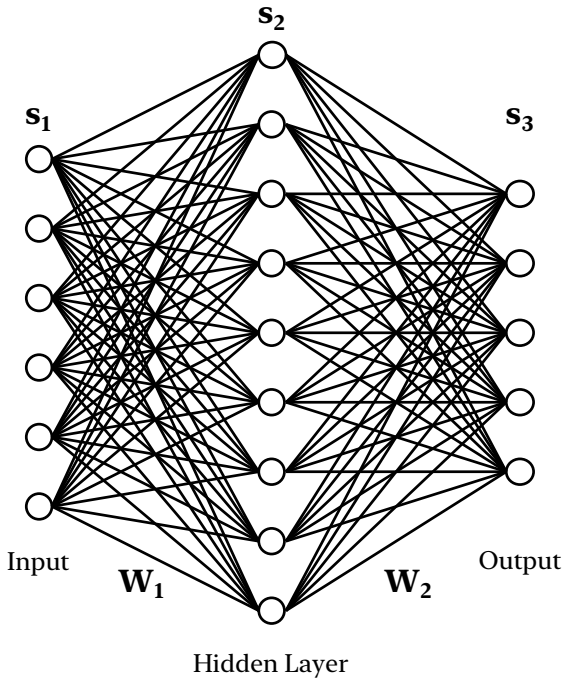



**Figure 2.2:** Multi-layer perceptron with a single hidden layer, six input neurons, nine neurons in the hidden layer and five output neurons. For a forward pass through the network the input states $\boldsymbol{s}_1$ are set to the input data. Then the McCulloch-Pitts dynamics are computed for the hidden layer as specified in equation 2.55 which produces the hidden states $\boldsymbol{s}_2$. At last the same procedure is carried out for the neurons in the last layer which produce the output states $\boldsymbol{s}_3$. The connections between the neurons visualize the weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, meaning this network consists of $54 + 45 = 99$ weights and $9 + 5 = 14$ biases.

When fitting a neural network to approximate an arbitrary function using empirical risk minimization one does not have access to a closed form solution of how to set the weights $\mathbf{W}_l$ and the bias terms $b_l$. This is the case because deep neural networks are compositions of non-linear functions which turns empirical risk minimization into a non-convex optimization problem. As a consequence, neural networks are trained using improved variants of stochastic gradient descent. This has the disadvantages that training a neural network often requires many iterations (gradient descent steps) and one cannot hope that the training converges to a specific set of parameters eventually. On the other hand, the stochastic gradient descent updates mean that the training is less likely to get stuck in a local minimum during the optimization process as the noise in the gradient estimates can cause a temporary increase in the expected loss [23]. To perform stochastic gradient descent the training data is divided into batches. For each batch consisting of $\tilde{m}$ samples the gradient of the prediction error $\mathcal{L}$ with respect to each parameter vector of the network is computed, resulting in the following update:

$$\mathcal{L} = \sum_{i=1}^{\tilde{m}} (\hat{f}(x_i) - y_i)^2 \tag{2.56}$$

$$\mathbf{W}_{(l,i)} \leftarrow \mathbf{W}_{(l,i)} - \alpha \nabla_{\mathbf{W}_{(l,i)}} \mathcal{L} \tag{2.57}$$

$$\mathbf{b}_l \leftarrow \mathbf{b}_l - \alpha \nabla_{\mathbf{b}_l} \mathcal{L} \tag{2.58}$$

The gradients for these updates are usually computed using the backpropagation algorithm, in which the error occurring in later layers is used to compute the error for layers closer to the input neurons. Hence the updates are computed in the opposite direction compared to the forward pass which produces a prediction. For details on backpropagation and how this is implemented in popular deep learning frameworks such as PyTorch [26], the interested reader may have a look at the corresponding chapter of [17].

# 3

# Learning from privileged time series information

## 3.1 Problem setting

In this section the exact problem that this thesis investigates shall be described mathematically. The different assumptions used in the theoretical analyses of later sections make use of the concepts introduced here.

**Problem structure.** Let $X_1, X_2, \ldots X_T$ with $X_t \in \mathcal{X} \subseteq \mathbb{R}^k$ be random variables that occur in the form of a time series, meaning that these variables are ordered chronologically and $X_t$ is observed at time $t \in \{1, 2, \ldots T\}$. Further, let $Y \in \mathcal{Y} \subseteq \mathbb{R}^q$ be called the outcome that can be regarded as the last element of the time series. In general we assume that $Y$ is a nonlinear function of $X_1$ with added Gaussian noise. In particular we represent the outcome as the composition of a nonlinear function $\Phi : \mathcal{X} \to \mathcal{Z} \subseteq \mathbb{R}^d$ and a linear function parameterized by $\theta \in \mathbb{R}^{d \times q}$ leading to the form of

$$Y = h(X_1) + \epsilon \quad \text{where} \quad h(\cdot) \coloneqq \theta^\top \Phi(X_1), \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2) \ . \tag{3.1}$$

Notice, that this structure provides a very general framework. In the course of this project the nonlinear function $\Phi$ will be referred to as the *representation function*. The task of interest is predicting the outcome $Y$ from baseline features $X_1$, while considering $X_2, X_3, \ldots X_T$ to be *privileged time-series information*, meaning this information is available only for learning but not for making predictions. Two particular kinds of data generating processes that fit this description, and that will be discussed in detail later, are illustrated in Figure 3.1.

In theoretical analysis, our work only considers the case where the time series is produced by a system which satisfies the Markov condition. This means that the conditional distribution of each variable $X_t$ given its predecessors has the following property:

$$p(X_t|X_{t-1}, X_{t-2}, ...X_1) = p(X_t|X_{t-1}) \tag{3.2}$$

Further, as we assume that $Y$ is either observable at time $T$ or thereafter, we also make a similar assumption about the outcome:

$$p(Y|X_t, X_{t-1}, X_{t-2}, ...X_1) = p(Y|X_t) \tag{3.3}$$

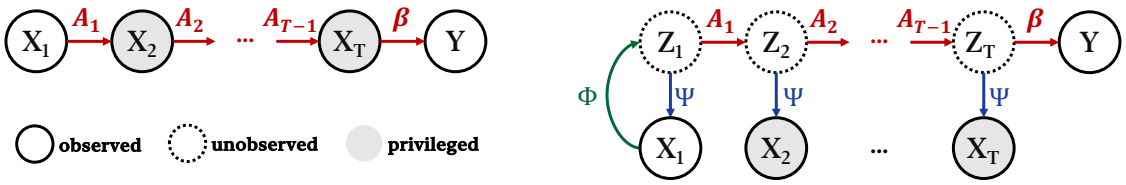



**Figure 3.1:** Linear-Gaussian data generating process on the left and the generalized version of a latent dynamical system on the right. $\Psi$ is the observation generating function, turning latent states $\{Z_t\}$ into observed variables $\{X_t\}$. The representation function $\Phi$ is the left inverse of $\Psi$.

For the joint distribution of all the random variables introduced above, this means that one gets the following factorization:

$$p(X_1, X_2, ...X_T, Y) = p(Y|X_T)\left[\prod_{t=2}^{T} p(X_t|X_{t-1})\right]p(X_1) \ . \tag{3.4}$$

Intuitively, the Markov assumption means in this particular setting that $X_t$ contains all relevant information about the future of the sequence that is given by $X_{t-1}$.

**Data sets, algorithms and risk.** Let $D$ denote a data set $D \in \mathcal{D}$ which contains $m \in \mathbb{N}_+$ samples. Each sample is one observed time series consisting of $T$ steps plus the outcome, meaning $D \coloneqq \{(x_{i,1}, x_{i,2}, ..., x_{i,T}, y_i)\}_{i=1}^{m}$. Data sets are drawn independently and identically distributed from an unknown joint distribution $p$ over all random variables of our time series.

Learning algorithms $\mathscr{A} : \mathcal{D} \to \mathcal{H}$ map data sets $D$ to a hypothesis $\hat{h}$. To make a prediction $\hat{y}$ the hypothesis is evaluated at inference time on a new test point $x_1$ such that $\hat{y} = \hat{h}(x_1)$. We are interested in the expected risk $\bar{R}_p$ produced by different algorithms as measured by a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$:

$$\bar{R}_p \coloneqq \mathbb{E}_D[R_p] \quad \text{with} \quad R_p \coloneqq \mathbb{E}_p[\mathcal{L}(\hat{h}(X_1), Y)] \quad \text{and} \quad \hat{h} = \mathscr{A}(D) \tag{3.5}$$

In other words, the risk describes the loss incurred on average over many data sets $D$ for a given algorithm $\mathscr{A}$. Throughout this project we consider the squared error loss $\mathcal{L}(y, y') \coloneqq \|y - y'\|_2^2$ for regression tasks to quantify the performance of different learning algorithms, as it is the goal to predict $Y$ from $X_1$.

An algorithm is considered *good* if it has low expected risk $\bar{R}_p$. In the regression tasks of this thesis this relates to low prediction error. We compare privileged learning algorithms $\mathscr{A}_P$ to classical learning algorithms $\mathscr{A}_C$. Privileged learners use samples of the joint distribution of $(X_1, X_2, \dots Y)$ during training, i.e. they make use of privileged time-series information. Classical learners on the other hand only observe samples of the form $(X_1, Y)$, which means they do not have access to privileged time-series information which makes this the common supervised learning approach. At test time both algorithms predict the outcome $Y$ *only* from $X_1$. This project seeks to identify conditions and algorithms for which the expected risk is lower when using

privileged information $\bar{R}_p(\mathscr{A}_P) < \bar{R}_p(\mathscr{A}_C)$. In other words we intend to find settings where privileged information is provably useful for making better predictions. When doing so, we compare algorithms that produce hypotheses $\hat{h}$ that are part of the same hypothesis class $\mathcal{H} \ni h$. This means not comparing linear estimators to non-linear estimators and not comparing different classes of non-linear estimators with each other. We analyze estimators of the form $\hat{h}(\cdot) = \theta^\top \Phi(\cdot)$ inspired by the problem definition of (3.1). We ensure the classical and privileged learners are of the same class by letting their representation functions $\hat{\Phi}_P$ (privileged) and $\hat{\Phi}_C$ (classical) share a hypothesis class, while combining each of the two with a linear estimator:

$$\hat{h}_P(\cdot) = \theta_P^\top \hat{\Phi}_P(\cdot) \quad \text{and} \quad \hat{h}_C(\cdot) = \theta_C^\top \hat{\Phi}_C(\cdot) \tag{3.6}$$

Consequently, differences in risk cannot be attributed to differences in approximation capability, instead they must be due to (i) information use, (ii) objective function and / or (iii) differences in optimization when no closed form solution is available.

In the following we introduce the data generating processes (DGPs) that will play an important role in the assumptions used for our theoretical results. The first is fully observed system using linear dynamics and the second is a nonlinear generalization of the first.

### 3.1.1 Linear-Gaussian system

In the following we describe linear-Gaussian systems as illustrated on the left side of Figure 3.1. Such a DGP produces time series consistent with the setting described in (3.1), while being fully observed. The system therefore contains no latent states and is made up of linear transitions and additive Gaussian noise. The baseline variable $X_1$ is sampled from an arbitrary distribution after which each subsequent variable $X_{t+1}$ is computed as a linear function of its predecessor $X_t$ plus Gaussian noise. At last the outcome is also a linear function of $X_T$. The previous work of Karlsson et al. [10] is based on assuming that data is generated by a linear-Gaussian system as described by the following structural equations.

**Definition 2** (Linear-Gaussian system)**.** *Let $X_1$ be drawn from an arbitrary distribution and let $\mathbf{A}_t \in \mathbb{R}^{k\times k}$ and $\boldsymbol{\beta} \in \mathbb{R}^{k\times q}$.*

$$\begin{aligned} X_{t+1} &:= A_t^\top X_t + U_{t+1} \quad \forall\ t\ \in \{1, 2,\ \ldots\ T-1\} \\ Y &:= \beta^\top X_T + U_y \\ U_t &\sim \mathcal{N}(0, \Sigma) \quad \forall\ t \in \{2\ \ldots\ , T, y\} \end{aligned}$$

$U_t$ denote independent noise variables with Gaussian distribution. The transitions are determined by matrices $A_t$ and $\beta$, while the noise is additive. As a consequence the expected outcome $\mathbb{E}[Y|X_1]$ can be described as a linear function of first random variable $X_1$,

$$\mathbb{E}[Y|X_1] = \beta^\top A_T^\top \ldots A_2^\top A_1^\top X_1\ . \tag{3.7}$$

One may also notice that every random variable of the system can be described as a linear function of an earlier variable in expectation. In the next subsection this framework is generalized into a latent dynamical system.

### 3.1.2 Latent linear-Gaussian system

The second system we introduce is the basis for most of the results of this project. The data generating process of a latent-linear Gaussian system is very similar to the linear-Gaussian system described above and is in fact a generalization of it. Its chain of linear transitions is hidden behind the observation generating function $\Psi$ which produces observed variables $\{X_t\}$ from hidden or latent variables $\{Z_t\}$. The chain of variables $Z_1, Z_2, \ldots, Z_T, Y$ is generated as a linear-Gaussian system as shown in Definition 2. The data generating process described by the definition below is illustrated on the right side of Figure 3.1.

**Definition 3** (Latent linear-Gaussian system)**.** *Let $Z_1$ be drawn from an arbitrary distribution and let $\Psi : \mathcal{Z} \to \mathcal{X}$ be an injective function with $\Phi : \mathcal{X} \to \mathcal{Z}$ its left inverse. With $A_t \in \mathbb{R}^{d\times d}$ and $\beta \in \mathbb{R}^{d\times q}$ let variables $Z_t$ be latent and $X_t$ be the observations generated as*

$$\begin{aligned} Z_{t+1} &:= A_t^\top Z_t + U_{t+1} \quad \forall\ t \in \{1, 2, \ldots T-1\} \\ X_t &:= \Psi(Z_t) \\ Y &:= \beta^\top Z_T + U_y \\ U_t &\sim \mathcal{N}(0, \Sigma) \quad \forall\ t \in \{2 \ldots, T, y\} \end{aligned}$$

The notation is changed here compared to the system in Definition 2 such that $X_t$ is always an observed variable, while $Z_t$ describes a latent variable. The data generated by such a system therefore comes in tuples of the same form $(X_1, X_2, \ldots, X_T, Y)$. Notice that in comparison to the fully observed linear-Gaussian system in Definition 2, here $E[Y|X_1]$ is not necessarily a linear function of $X_1$. This now depends on the choice of the observation generating function $\Psi$. We assume it is an *injective* function, meaning it has a left inverse $\Phi : \mathcal{X} \to \mathcal{Z}$, such that

$$z = \Phi(\Psi(z)) \ \forall\ z \in \mathcal{Z}\ . \tag{3.8}$$

$\Psi$ being injective means that all information in $Z_t$ is also contained in $X_t$. In other words, $Z_t$ can be reconstructed from $X_t$ if one knows $\Phi$. We refer to $\Phi$ as the representation function as it reveals the unobserved representation $z_t$ that produced a particular data point $x_t$. The latent system is a generalization of the simpler linear-Gaussian system as the two become identical when $\Psi(z) = z$. Further, it is easy to show that when $\Psi$ is a linear function the latent dynamical system one obtains is in fact also a linear-Gaussian system. This highlights that Definition 3 provides a more general framework compared to Definition 2.

Latent dynamical systems similar to the ones presented here have proven successful at modelling a variety of phenomena as for example fluid dynamics in physics [27] and human brain activity in neuroscience [28]. In the next section, we assume that data is generated from a latent linear-Gaussian system and that its representation function $\Phi$ is known up to a linear transform.

## 3.2 Learning with true representations known

In this section we present the generalized LuPTS algorithm and derive it from previous work on learning from privileged time-series information. We prove its improved sample efficiency compared to the classical learning alternative on latent dynamical systems as presented in Definition 3 for the case when the true representation function of the data generating process is known up to a linear transform. While doing so, we notice a phase transition in the generalized LuPTS estimator, that lets the benefit of privileged information vanish when the number of features is higher than the number of samples. At last, we reason about the bias and variance characteristics of generalized LuPTS and its use with popular kernels.

**Classical learning with OLS.** Before introducing generalized LuPTS, we present the classical learning alternative that this new algorithm will be compared to. Let $\hat{\Phi} : \mathcal{X} \to \hat{\mathcal{Z}} \subseteq \mathbb{R}^{\hat{d}}$ be an estimate of the true representation function $\Phi$ as introduced in Definition 3. If the true representation function $\Phi$ is known, i.e. $\hat{\Phi}(\cdot) = \Phi(\cdot)$, then using an ordinary least squares estimator on the inferred latent states $\hat{Z}_1 \in \hat{\mathcal{Z}} \subseteq \mathbb{R}^{\hat{d}}$ will result in the minimum variance unbiased estimator that does not make use of privileged information. In other words, the classical learner applies the estimated representation function $\hat{\Phi}$ to the baseline features $X_1$ and uses OLS linear regression afterwards. Let $\hat{\mathbf{Z}}_t \in \mathbb{R}^{m \times \hat{d}}$ be the design matrix containing the inferred latent states $\hat{Z}_t = \hat{\Phi}(X_t)$ of time step $t$ for $m$ training samples, then the estimator $\hat{h}_C$ takes the following form, analogous to (2.24):

$$\hat{h}_C(\cdot) \coloneqq \theta_C^\top \hat{\Phi}(\cdot) \quad \text{with} \quad \theta_C \coloneqq (\hat{\mathbf{Z}}_0{}^\top \hat{\mathbf{Z}}_0)^\dagger \hat{\mathbf{Z}}_0{}^\top \mathbf{Y} \tag{3.9}$$

In the coming section the privileged learning alternative to this classical learning algorithm is presented.

### 3.2.1 Generalized LuPTS

To predict $Y$ from $X_1$ generalized LuPTS infers the latent state $\hat{Z}_1 = \hat{\Phi}(X_1)$, then simulates the linear transition dynamics of the latent linear-Gaussian system in a stepwise fashion before producing its estimate of the outcome. This is visualized on the right pane of Figure 3.2. During training, generalized LuPTS infers all latent states $\{\hat{Z}_t\}$ from $\{X_t\}$ using the estimated representation function and minimizes the error for each transition individually, i.e. it fits the OLS estimators $\{\hat{A}_t, \hat{\beta}\}$ to model each transition of the time series in the latent space implied by $\hat{\Phi}$. One can therefore view generalized LuPTS as minimizing the following objective

$$\min_{\hat{A}_1, \hat{A}_2, \ \dots \ \hat{A}_{T-1}, \hat{\beta}} \quad \sum_{i=1}^{m} \left[ \sum_{t=1}^{T-1} \| \hat{A}_t^\top \hat{\Phi}(x_t^{(i)}) - \hat{\Phi}(x_{t+1}^{(i)}) \|_2^2 \right] + \| \hat{\beta}^\top \hat{\Phi}(x_T^{(i)}) - y^{(i)} \|_2^2 \ , \tag{3.10}$$

where $x_t^{(i)}$ is time step $t$ of the $i$-th training example. Notice, that because $\hat{\Phi}$ is fixed, the objective is separable between the different time steps, meaning every linear estimator $\hat{A}_t$ or $\beta$ can be fit individually using ordinary least squares. The left

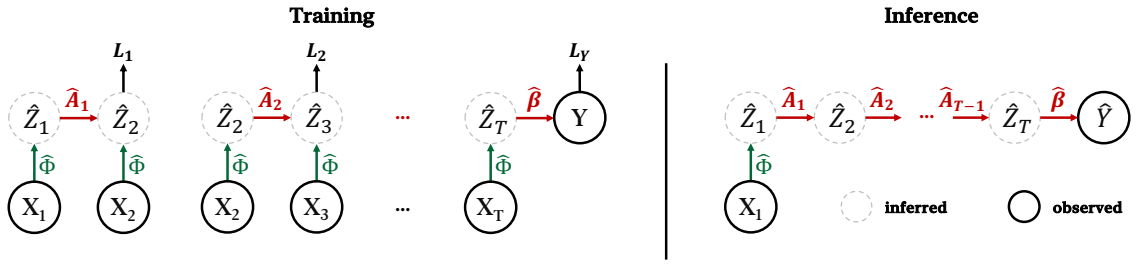



**Figure 3.2:** Visualization of the stepwise estimation approach of generalized LuPTS (Algorithm 1). Parameters $\{\hat{A}_t, \hat{\beta}\}$ are fit independently, minimizing the single step squared error (losses $L_t$) using ordinary least squares. During inference the latent dynamics are simulated from start to end to estimate the outcome.

pane of Figure 3.2 illustrates the single step losses in the latent space during training.

In the following the generalized LuPTS algorithm is presented in two versions: The first uses an explicit fixed feature map $\hat{\Phi}$, while the second uses a reproducing kernel $\kappa(x, x') := \langle \hat{\Phi}(x), \hat{\Phi}(x') \rangle$ and therefore only applies the feature map implicitly. In the following matrices written in bold letters are the design matrices $\hat{Z}_t \in \hat{\mathcal{Z}} \subseteq \mathbb{R}^{m \times \hat{d}}$ consisting of $m$ samples arranged along the rows and $\hat{d}$ features on the columns. We denote the dimensionality of the true latent variables $Z_t$ with $d$, while $\hat{d}$ refers to the dimension of the estimated latent variables $\hat{Z}_t$ implied by $\hat{\Phi}$. These spaces might be of different dimensionality. We may now present the generalized LuPTS algorithm.

**Algorithm 1:** Generalized LuPTS

**Input:** Data $D = (\{\mathbf{X}_t\}, \mathbf{Y})$ with $\mathbf{X} \in \mathbb{R}^{m \times k}$, $\mathbf{Y} \in \mathbb{R}^{m \times q}$,
Representation $\hat{\Phi} : \mathcal{X} \to \hat{\mathcal{Z}}$ or kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

**if** using a fixed representation $\hat{\Phi}$ **then**

$\hat{\mathbf{Z}}_t = [\hat{\Phi}(x_{1,1}), ..., \hat{\Phi}(x_{m,1})]^\top$ for $t = 1, ..., T$

$$\hat{\theta}_{\mathrm{P}} := \Big[ \prod_{t=1}^{T-1} \underbrace{(\hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_t)^\dagger \hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_{t+1}}_{\hat{A}_t} \Big] \underbrace{(\hat{\mathbf{Z}}_T^\top \hat{\mathbf{Z}}_T)^\dagger \hat{\mathbf{Z}}_T^\top \mathbf{Y}}_{\hat{\beta}}$$

$\hat{h}_{\mathrm{P}}(\cdot) := \hat{\theta}_{\mathrm{P}}^\top \hat{\Phi}(\cdot)$

**else if** using kernel $\kappa$ **then**

$\hat{\mathbf{K}}_t = \kappa(\mathbf{X}_t, \mathbf{X}_t)$ for $t = 1, ..., T$

$\boldsymbol{\alpha} := \hat{\mathbf{K}}_1^\dagger \left[ \prod_{t=2}^{T} \hat{\mathbf{K}}_t \hat{\mathbf{K}}_t^\dagger \right] \mathbf{Y}$

$\hat{h}_{\mathrm{P}}(\cdot) := \sum_{i=1}^{n} \alpha_i \kappa(x_{i,1}, \cdot)$

**return** $\hat{h}_{\mathrm{P}}$

Note that while generalized LuPTS is constructed with latent dynamical systems in mind, it is a generalization of the linear LuPTS algorithm presented by Karlsson et al [10]. Latent linear-Gaussian processes like described in Definition 3 become linear-Gaussian systems when the observation generating function is also linear $x = \Psi(z) = \mathbf{M}z$ for some matrix $\mathbf{M}$. The same logic applies to generalized LuPTS: It becomes Karlsson's linear LuPTS whenever the representation function $\hat{\Phi}$ is linear, which makes Algorithm 1 a generalization.

**Generalized LuPTS with kernels.** After the introduction of Algorithm 1, its kernel version shall be derived from the alternative of using a fixed feature map. The hypothesis $h_P(\cdot) = \theta_P^\top \hat{\Phi}(\cdot)$ returned by generalized LuPTS makes use of the parameter

$$\hat{\theta}_{\text{P}} \coloneqq \left[ \prod_{t=1}^{T-1} \underbrace{(\hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_t)^\dagger \hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_{t+1}}_{\hat{A}_t} \right] \underbrace{(\hat{\mathbf{Z}}_T^\top \hat{\mathbf{Z}}_T)^\dagger \hat{\mathbf{Z}}_T^\top \mathbf{Y}}_{\hat{\beta}} , \tag{3.11}$$

which is a composition of the single step OLS estimators (compare to Figure 3.1). To transform this into the kernel variant one takes two steps. First one uses the simple matrix identity from equation (2.47) in order to reveal Gram matrices $\hat{\mathbf{Z}}_t \hat{\mathbf{Z}}_t^\top$ of the estimated latent states of different time steps $t$. In the next step the Gram matrices are replaced by the kernel matrices $\mathbf{K}_t$ corresponding to the kernel $\kappa$,

$$\begin{aligned}
\hat{\theta}_{\text{P}} &= \left[ \prod_{t=1}^{T-1} (\hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_t)^\dagger \hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_{t+1} \right] (\hat{\mathbf{Z}}_T^\top \hat{\mathbf{Z}}_T)^\dagger \hat{\mathbf{Z}}_T^\top \mathbf{Y} \\
&= \left[ \prod_{t=1}^{T-1} \hat{\mathbf{Z}}_t^\top (\hat{\mathbf{Z}}_t \hat{\mathbf{Z}}_t^\top)^\dagger \hat{\mathbf{Z}}_{t+1} \right] \hat{\mathbf{Z}}_T^\top (\hat{\mathbf{Z}}_T \hat{\mathbf{Z}}_T^\top)^\dagger \mathbf{Y} \\
&= \hat{\mathbf{Z}}_1^\top (\hat{\mathbf{Z}}_1 \hat{\mathbf{Z}}_1^\top)^\dagger \hat{\mathbf{Z}}_2 \hat{\mathbf{Z}}_2^\top (\hat{\mathbf{Z}}_2 \mathbf{Z}_2^\top)^\dagger \hat{\mathbf{Z}}_3 \mathbf{Z}_3^\top \ldots \hat{\mathbf{Z}}_T \hat{\mathbf{Z}}_T^\top (\hat{\mathbf{Z}}_T \hat{\mathbf{Z}}_T^\top)^\dagger \mathbf{Y} \\
&= \hat{\mathbf{Z}}_1^\top \mathbf{K}_1^\dagger \mathbf{K}_1 \mathbf{K}_1^\dagger \mathbf{K}_2 \mathbf{K}_2^\dagger \ldots \mathbf{K}_T \mathbf{K}_T^\dagger \mathbf{Y} \\
&= \hat{\mathbf{Z}}_1^\top \mathbf{K}_1^\dagger \left[ \prod_{t=1}^{T} \mathbf{K}_t \mathbf{K}_t^\dagger \right] \mathbf{Y} \ .
\end{aligned}$$

Now one may define $\boldsymbol{\alpha} \coloneqq \hat{\mathbf{K}}_1^\dagger \left[ \prod_{t=2}^{T} \hat{\mathbf{K}}_t \hat{\mathbf{K}}_t^\dagger \right] \mathbf{Y}$ in order to use $\hat{h}_{\text{P}}(\cdot) \coloneqq \sum_{i=1}^{n} \alpha_i \kappa(x_{i,1}, \cdot)$ for a new prediction. Notice that this form is exactly the same as for kernelized OLS as shown in Section 2.6. Given the same kernel $\kappa$, the only difference between these two estimators is how $\boldsymbol{\alpha}$ is computed. This highlights again that these two learning algorithms produce hypotheses in the same class.

We may now state our main Theorem about generalized LuPTS. It says that when data is generated by a latent linear-Gaussian system with the true representation function known up to a linear transform, generalized LuPTS is never worse in terms of sample efficiency compared to classical learning for any sample size in expectation over data sets.

**Theorem 1.** *Let $D$ be a data set drawn from a distribution $p$, as produced by a dynamical system as described in Definition 3. Assume that the left inverse $\Phi : \mathcal{X} \to \mathcal{Z}$ of the observation function $\Psi$ is known up to a linear transform, explicitly or through a kernel $\kappa(x, x') = \langle \hat{\Phi}(x), \hat{\Phi}(x') \rangle$, i.e., there exists a matrix $B$ with linearly independent columns such that $\hat{\Phi}(x) = B\Phi(x)$ with $\hat{\Phi} : \mathcal{X} \to \hat{\mathcal{Z}}$ for all $x \in \mathcal{X}$. Then, it holds for the privileged learner $\mathscr{A}_{\text{P}}(D) = \hat{h}_{\text{P}}$ (generalized LuPTS of Algorithm 1) and the classical learner $\mathscr{A}_C(D) = \hat{h}_C$ (3.9),*

$$\overline{R}(\mathscr{A}_{\text{P}}) = \overline{R}(\mathscr{A}_C) - \mathbb{E}_{\hat{h}_{\text{P}}, X_1}[\text{Var}_D(\hat{h}_C(X_1) \mid \hat{h}_{\text{P}})] \ . \tag{3.12}$$

*Proof sketch.* First we show in Lemma 1 that generalized LuPTS and the classical learner make the same predictions if the learned representation is linearly close (as given in the assumption) to the true representation. Then we consider three different cases for the dimensionality of the true latent system $d$ and the sample size $m$: (i) When $m = d$ the kernel matrices will be invertible, yielding the result of Proposition 1, which means the risk is the same for both algorithms. (ii) When $m < d$ the technical assumptions required for the theorem statement cannot be fulfilled. (iii) When $m > d$ we can use Lemma 1 to reason about an estimator using the true map $\Phi$ and for this latter estimator Theorem 1 of Karlsson et al. [10] holds, which gives the desired result. A full proof is given in Appendix A.1. □

Theorem 1 implies that Algorithm 1 will accrue lower or equal risk in expectation over data sets than the classical learner since $\mathrm{Var}_D(\cdot) \geq 0$. The assumptions about the data generation process are restrictive: Not every function for which $Y = \theta^\top \Phi(X_1)$ additionally obeys $X_t = \Psi(Z_t)$ for all time steps. However, our assumptions on the data generating process are much more general than the results of Karlsson et al. [10], who make a similar statement for linear Gaussian-systems and linear estimators. Firstly, our result extend the LuPTS approach to nonlinear estimation through either feature maps or kernels. Secondly, one may use the kernel variant on objects that are not vectors in $\mathbb{R}^k$. All that is required is having access to a kernel $\kappa$ that can act as a similarity measure for two objects.

At last, we also extended LuPTS to the underdetermined case by using the Moore-Penrose pseudoinverse [12]. When doing so, we notice that generalized LuPTS undergoes a phase transition with exploding variance when the number of features $\hat{d}$ become larger than the number of samples $m$, after which the benefit of privileged information vanishes and generalized LuPTS becomes equivalent to the corresponding classical learner. This is the statement of the following proposition.

**Proposition 1.** *Let $\hat{\Phi} : \mathcal{X} \to \hat{\mathcal{Z}} \subseteq \mathbb{R}^{\hat{d}}$ be any map with corresponding kernel $\kappa$. Let $\hat{\mathbf{K}}_t$ be the Gram matrix of $\kappa$ applied to $\mathbf{X}_t$ and let $\hat{h}_C, \hat{h}_P$ be classical* (3.9) *and privileged (Algorithm 1) estimates. Then,*

$$\hat{\mathbf{K}}_t \text{ is invertible for all } t \implies \hat{h}_P = \hat{h}_C.$$

*$\hat{\mathbf{K}}_t$ is noninvertible whenever $m > \hat{d}$, assuming linearly independent features.*

*Proof sketch.* When the Gram matrices $\mathbf{K}_t$ are invertible, the pseudo-inverse coincides with the inverse, and factors $\mathbf{K}_t \mathbf{K}_t^\dagger$ in the generalized LuPTS estimator cancel, making it equal to the classical learner. □

Proposition 1 says that the gap in risk between privileged and classical learning closes when one has more features $\hat{d}$ than samples $m$. However this is only a statement about generalized LuPTS, it does not state that no better privileged learner exists. Figure 3.3 illustrates the behaviour of generalized LuPTS and the classical learner when increasing the number of features $\hat{d}$ for a fixed sample size $m$. The double descent behaviour exhibited by our algorithm in Figure 3.3 is familiar from previous analyses on linear estimators [29]. The variance of both estimators increases as the number of features approaches the point $\hat{d} = m$. When the variance

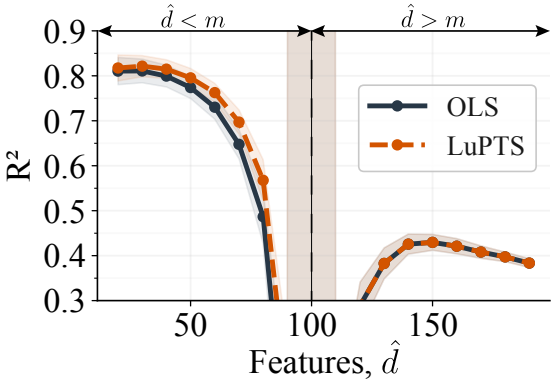



**Figure 3.3:** The two regimes of generalized LuPTS. Shown is the average coefficient of determination $R^2$ (higher is better), which is proportional to the empirical squared error risk for two learning algorithms and varying feature counts over 50 data generating processes as described in Definition 2. When the number of features $\hat{d}$ is smaller than the number of samples $m = 100$ and the features are linearly independent, one gets lower risk on average compared to classical learning according to Theorem 1. However, when $\hat{d} > m$ generalized LuPTS and the classical learner (OLS) become equivalent as shown by Proposition 1.

reduces for a second time, generalized LuPTS and its corresponding classical learner obtain the same risk.

Theorem 1 requires that the representation function is known up to a linear transform. In practical problems where the data generating process is unknown, this cannot be expected. As a consequence it is of great practical interest to explore the use of generalized LuPTS with kernels or feature maps that work well for unknown representation functions. This is discussed in detail in the following section.

## 3.3 Random feature maps for unknown representations

As generalized LuPTS can be used with any feature map or kernel, universal kernels come to mind for the case where the representation function $\Phi$ which connects observations $\{X_t\}$ to latent variables $\{Z_t\}$ is unknown. Universal kernels have dense reproducing kernel Hilbert spaces, which allow for the approximation of any continuous function [30]. The idea of combining generalized LuPTS with such a kernel is appealing as one could approximate any unknown representation function. A popular example of a universal kernel is the Gaussian radial basis function kernel $\kappa(x, x') = \exp\left(\| - \frac{x-x'}{2\sigma^2}\|_2^2\right)$. It is viewed as a generalization of the polynomial kernel with infinitely many basis functions that make up its associated feature map, thus its corresponding RKHS has infinite dimensionality. An issue arises from combining universal kernels with generalized LuPTS: It can be shown that universal kernels always produce positive definite and thus invertible kernel matrices [31]. Proposition 1 tells us, that this leads to generalized LuPTS becoming equivalent with the classical

learner using the same kernel. As a consequence, we cannot expect any advantage from using privileged time-series information when combining generalized LuPTS with universal kernels.

**Random features as an approximation of universal kernels.** As a consequence, we look to random feature methods as an alternative to universal kernels, as these methods can be seen as approximating the RKHS of universal kernels. The idea behind these approaches can be briefly described by sampling a random matrix $W$ that is multiplied with variates $x$ before applying an elementwise nonlinear function. Using those new features one can then fit a linear model. The most popular example of this are random Fourier features introduced by Rahimi et al. in 2007 [32]. The authors show that one can approximate any arbitrary continuous function with this approach if only given enough features and samples. Random Fourier features (RFF) are computed as

$$\hat{\Phi}^{\gamma}_{\mathrm{RFF}}(x) = \sqrt{2/\hat{d}}\left[\cos(\sqrt{2\gamma}W_{\mathcal{N}}^{\top}x + b)\right] \ . \tag{3.13}$$

The number of random features is denoted by $\hat{d}$, meaning $W_{\mathcal{N}} \in \mathbb{R}^{k\times\hat{d}}$, while $b \in \mathbb{R}^{\hat{d}}$ describes a bias term for each random feature. The cosine function is applied elementwise and the elements of $W_{\mathcal{N}}$ are sampled from a Gaussian distribution $(W_{\mathcal{N}})_{i,j)} \sim \mathcal{N}(0,1)$. The biases are sampled from a uniform distribution $b_i \sim \mathcal{U}(0, 2\pi)$. $\gamma$ is a bandwidth hyperparameter that just like the number of random features needs to be chosen manually. After sampling the features one would then fit the parameter $\theta$ of a linear model to obtain the estimator $\hat{h}(\cdot) = \theta^{\top}\hat{\Phi}^{\gamma}_{\mathrm{RFF}}(\cdot)$.

In similar work, Sun et al. [33] present random ReLU features (RRF) which also have universal approximation capabilities and make use of the rectified linear unit function $f_+(\cdot) = \max(0, \cdot)$. Random ReLU features are computed with slightly different sampling. The random weights are sampled uniformly $(W_{\mathcal{U}})_{i,j} \sim \mathcal{U}(-1, 1)$ with $W_{\mathcal{U}} \in \mathbb{R}^{k\times\hat{d}}$ and also here $\gamma$ plays the role of a bandwidth hyperparameter. Let $[x, y]$ denote the concatenation of vectors $x$ and $y$, then random ReLU features are given by

$$\hat{\Phi}^{\gamma}_{\mathrm{RRF}}(x) = f_+(\gamma W_{\mathcal{U}}^{\top}[x; 1]) \ . \tag{3.14}$$

While universal kernels produce kernel matrices $\mathbf{K}_t$ that are always invertible and thus render generalized LuPTS equivalent to classical learning, combining random features with generalized LuPTS allows us to control the number of features $\hat{d}$ making sure $\hat{d} < m$. As a consequence, we get the advantage of the universal approximation capabilities of random features *and* the variance reduction associated with generalized LuPTS. We point out, that the variance reduction promised by Theorem 1 only holds for the case where the representation function is known up to linear transformation, however one may hope to find such a representation with the use of random features given enough samples and features. Next, we justify the use of random features with generalized LuPTS by showing that this algorithm can be

made universally consistent.

**Consistency of generalized LuPTS with random features.** Consistency is a desirable property of estimators and describes the notion that an estimator converges in probability to the true function in the limit of infinite samples $m$. Sun et al. [33] show that random features $\hat{\Phi}$ combined with a linear model $\hat{h}(\cdot) = \theta^\top \hat{\Phi}(\cdot)$ can approximate any noiseless continuous function $f : \mathcal{X} \to \mathcal{Y}$ up to arbitrary precision in the limit of samples $m$ and features $\hat{d}$. We apply the same reasoning to generalized LuPTS to state: *Generalized LuPTS with a random feature map can be made a universally consistent estimator of* $h(x_1) = \mathbb{E}[Y|x_1]$ *under appropriate assumptions.* For a precise statement and a proof see Appendix A.2. As universal consistency describes the asymptotic behaviour in the limit of infinite samples and random features, this offers only limited insight into the benefits of privileged information in small sample settings, where performance will be a bias-variance trade-off.

**Variance reduction & bias amplification.** Generalized LuPTS is only guaranteed lower variance compared to the classical estimator when data is produced by a latent-linear Gaussian system with its representation function known up to a linear transform, see Theorem 1. Despite having theory only for this case, our empirical results (Section 4) suggest this applies more widely, as we have never seen an example of privileged time-series information use resulting in a bias increase. When $\hat{\Phi}$ is a bad approximation of $\Phi$ the classical and the privileged learners used in this chapter will be biased. Next we demonstrate that LuPTS may amplify this bias, meaning that it increases with the number of privileged time points, compared to classical learning. We show this theoretically in Appendix A.3 and demonstrate it empirically here.

Figure 3.4 illustrates the prediction accuracy of different variants of generalized LuPTS trained and tested on data from latent-linear Gaussian systems with varying sequence lengths. For experiment details see Appendix A.4. Linear LuPTS performs worse than the classical learner (OLS) as the observation generating function $\Psi$ is not linear here, in other words linear LuPTS is strongly biased and Theorem 1 does not hold for it. If linear LuPTS was unbiased, meaning data is produced by a linear dynamical system as in Definition 2, we would expect the privileged learner to outperform the classical learner. Here, bias is amplified as the sequence length increases, which lets the gap between the classical and the privileged learner grow. For the random feature variants of Section 3.3, each privileged learner performs better compared to its classical counterpart even for long sequences which highlights that these methods exhibit little bias. In summary there are two opposing effects at play here: If generalized LuPTS is biased, i.e. the representation $\hat{\Phi}$ is bad, the stepwise prediction approach amplifies this bias, while it appears that it always results in lower variance. Whether our privileged learner is still preferable to classical learning in terms of prediction risk appears to depend on the amount of bias that gets amplified. Our experiments imply the variance reduction mostly dominates when using random features, whereas this is not always the case for linear LuPTS.

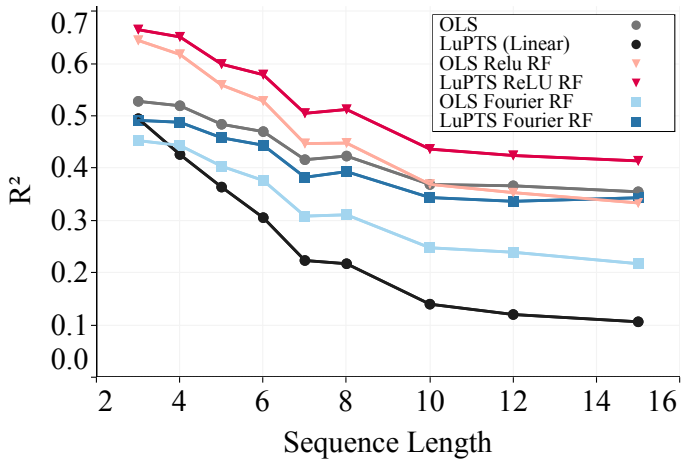



**Figure 3.4:** Different variants of Algorithm 1 and the classical learning alternative (3.9) applied on data sets with varying sequence lengths generated by **Square-Sign**($d = 10$, $q = 3$) which is latent-linear Gaussian system as described by Definition 3. We use $m = 1000$ samples and report the mean coefficient of determination $R^2$ over 500 repetitions. Notice that for the linear models in (gray and black) the classical learner outperforms the privileged learner with this gap increasing with $T$. The nonlinear models (colored) do not follow this pattern.

The phenomenon of bias amplification is familiar from e.g., model-based and model-free reinforcement learning [34], where a longer roll out horizon amplifies the bias induced from using a model, while using no model results in high variance. As the bias with random features may still be high for small sample settings as will be seen in the experiments of Chapter 4, we next present privileged representation learning algorithms to trade off bias and variance more efficiently.

## 3.4 Privileged time series representation learning

Up until now the representation $\hat{\Phi}$ was considered fixed, either because $\Phi$ was known up to a linear transform (explicitly or implicitly) or because of the use of random feature methods. In practical problems $\Phi$ cannot be expected to be known in general and as argued above random feature methods may still suffer from high bias in small sample settings. As a consequence, this section is inspired by marrying the ideas behind generalized LuPTS with the expressiveness of deep representation learning. To demonstrate how generalized LuPTS is related to the algorithms that follow in this Section, we highlight that Algorithm 1 produces minimizers $\{\hat{A}_t\}, \hat{\beta}$ of the following objective with respect to $\alpha = \{\{\hat{A}_t\}, \hat{\beta}\}$ with fixed $\hat{\Phi}$

$$\mathcal{L}_{\text{SRL}}(\alpha) := \frac{1}{NT} \sum_{i=1}^{N} \left[ \sum_{t=1}^{T-1} \frac{1}{\hat{d}} \left\| \hat{A}_t^\top \hat{\Phi}(x_{i,t}) - \hat{\Phi}(x_{i,t+1}) \right\|_2^2 + \frac{1}{q} \left\| \hat{\beta}^\top \hat{\Phi}(x_{i,T}) - y_i \right\|_2^2 \right] . \quad (3.15)$$

Recall that this is an equivalent loss to the formulation given in (3.10). Objective (3.15) and the systems described by Definition 3 lend themselves to methods

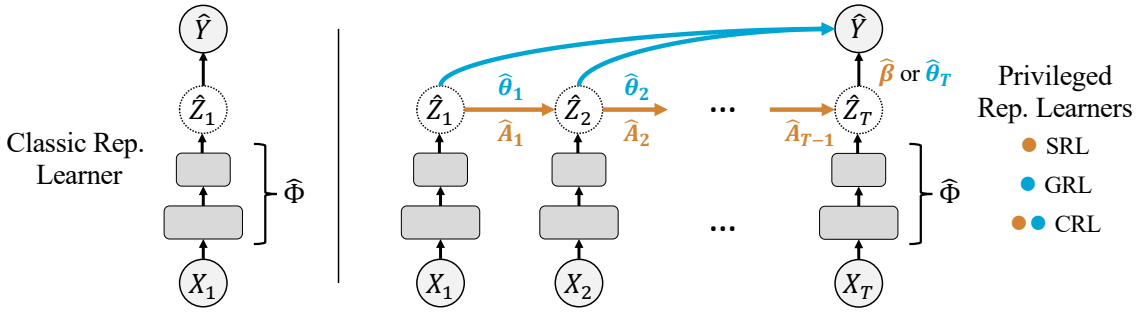



**Figure 3.5:** Classical (left) and privileged (right) representation learners. $\hat{\Phi}$ is shared across all time steps. GRL models the direct maps $\hat{\theta}_t$ to the outcome; SRL models the single steps $\hat{A}_t$ and $\hat{\beta}$. CRL combines the two.

which also learn the representation $\Phi$ in addition to the latent dynamics $\{A_t, \beta\}$. Next, we present three representation learning algorithms, which do exactly this. All learners that we compare as part of the same hypothesis class use equivalent encoders, parameterized by neural networks, to represent $\hat{\Phi}(\cdot)$ and linear layers to model the relations between the latent variables $\{Z_t\}$ and the outcome $Y$. The classical learner predicts the outcome linearly from $\hat{Z}_1$. All architectures that we present in the following are visualized jointly in Figure 3.5.

**SRL.** The first privileged representation learner directly optimizes objective (3.15), just like generalized LuPTS, *but now also learning the representation* $\hat{\Phi}$, parameterized by a neural network. We refer to this model as *stepwise representation learner* (SRL). As we will see in experiments, a drawback of this approach is that representations may favor predicting transitions $\hat{z}_{i,t} \to \hat{z}_{i,t+1}$ with small error, while losing information relevant for the target outcome in the process. At test time, for a new input $x_1$, SRL composes the stepwise dynamics to output $\hat{h}_{\text{P}}(x_1) = \hat{\beta}^\top \hat{A}_T^\top \dots \hat{A}_1^\top \hat{\Phi}(x_1)$. In essence, SRL is closely related to LuPTS but instead of using a fixed representation with OLS estimators for the single step transitions, it uses a learned representation and does not make use of a closed form solution.

**GRL.** To avoid the information loss of SRL, we consider its conceptual opposite, using privileged time series information *only* to predict the outcome. To do this, a linear output layer $\hat{\theta}_t^\top \hat{Z}_t$ is used to predict $Y$ at every time step $t$. Recall that, in the latent dynamical system of Definition 3, the expected outcome is linear in the latent state at *any* time step. This second representation learning method is related to multi-view learning, in which predictions of the same quantity are made from multiple "views" [35]. Here, we make predictions of $Y$ from $T$ different time steps during training, while only having access to one perspective at test time. We dub the model *greedy privileged representation learner* (GRL) due to its focus on the outcome. It minimizes the objective

$$\mathcal{L}_{\text{GRL}}(\hat{\Phi}, \{\hat{\theta}_t\}) := \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} w_t \left\| \hat{\theta}_t^\top \hat{\Phi}(x_{i,t}) - y_i \right\|_2^2. \tag{3.16}$$

During inference this algorithm returns $\hat{h}_{\mathrm{P}}(\cdot) = \hat{\theta}_1^\top \hat{\Phi}(\cdot)$. We introduce an additional hyperparameter $\lambda \in (0, 1)$ to place more weight on the loss term that is relevant at inference time. As a consequence, we choose $w_t = \lambda$ for $t = 1$ and $w_t = 1 - \lambda$ otherwise. We expect GRL to have less bias than SRL, but higher variance since less structure is imposed on the representation $\hat{\Phi}$.

**CRL.** Finally, we combine the previous two approaches (SRL and GRL), modelling all latent linear transitions as well as the linear outcome function for every time step. We introduce a hyperparameter $\lambda \in [0, 1]$ to trade off the two types of losses and arrive at the *combined representation learner* (CRL). With $\alpha = (\hat{\Phi}, \{\hat{A}_t\}, \{\hat{\theta}_t\})$ the entire parameter vector, CRL minimizes the objective

$$\mathcal{L}_{\mathrm{CRL}}(\alpha) := \frac{\lambda}{NTq} \sum_{i,t} \left\| \hat{\theta}_t^\top \hat{\Phi}(x_{i,t}) - y_i \right\|_2^2 + \frac{1-\lambda}{N(T-1)\hat{d}} \sum_{i,t} \left\| \hat{A}_t^\top \hat{\Phi}(x_{i,t}) - \hat{\Phi}(x_{i,t+1}) \right\|_2^2. \tag{3.17}$$

We make test-time predictions using $\hat{h}_{\mathrm{P}}(x_1) = \hat{\theta}_1^\top \hat{\Phi}(x_1)$, which is exactly the same as in GRL for simplicity. This is not the only possible choice as one could combine different linear estimators to get from $\hat{Z}_1$ to $\hat{Y}$, essentially choosing a path along different arrows in the right panel of Figure 3.5. One may also view GRL as a special case of CRL as they become equivalent for $\lambda = 1$ in (3.17) and (3.16). In the coming chapter we analyze the representation learning algorithms introduced here together with different variants of generalized LuPTS empirically.

# 4
# Experiments

We compare classical learning to variants of generalized LuPTS (Algorithm 1) and the privileged representation learners of Section 3.4 on two synthetic and three real-world time-series data sets empirically. The former are designed to satisfy the assumption of a latent linear dynamical system (Definition 3) with a nonlinear observation function. The latter are time series data sets where one could reasonably hope that privileged time-series information would be helpful for making more accurate predictions. We (i) verify our theoretical findings by analyzing the sample efficiency and bias-variance characteristics of the given algorithms; (ii) demonstrate that generalized LuPTS with random features succeeds in settings where linear LuPTS suffers from large bias; (iii) point out that privileged representation learners offer even greater sample efficiency in practice (all in Section 4.3) and (iv) study how well these algorithms recover the true latent variables $\{Z_t\}$ and how this relates to predictive accuracy (Section 4.4). At first we give some insights on how the experiments described in this section were conducted.

## 4.1 Experiment setup

All prediction tasks that models are tested on in the course of this chapter are regression problems and we report the mean coefficient of determination ($R^2$), proportional to the squared-error risk $\overline{R}$, for varying sample sizes, sequence lengths and prediction horizons. For a given data set, we select a combination of training set sizes and sequence length. For each unique combination of these parameters the models of interest are trained repeatedly with different random sampling. For each repetition the data is split into a train and a test set randomly before hyperparameter tuning (using cross-validation) is performed. Next, a given model is retrained on the full set of training data of the selected sample size. At last, each model's predictions on the test set are scored using the coefficient of determination $R^2$. On synthetic data the test set contains 1000 samples. In the case of real-world data, where samples are limited, we test on 20% of all available data. Every model uses standardized data for training and inference. Hyperparameter tuning is carried out using random search with five-fold cross-validation in every repetition. The preprocessing used for real-world data and the generation procedure of synthetic data is unique to each data set.

The algorithms considered in the following are divided into two groups. The first group comprises generalized LuPTS with the linear kernel (LuPTS) and the two random feature maps shown in Section 3.3: Random Fourier features (Fourier RF) [32]

and random ReLU features (ReLU RF) [33]. The classical learners for this group are OLS estimators used with the same kernel or feature map. The second group consists of the representation learners SRL, GRL and CRL, as well as the corresponding classical representation learner (Classic Rep.). For tabular data, their encoder is a multi-layer perceptron with three hidden layers of 25 neurons each, producing 25 dimensional latent states $\{\hat{Z}_t\}$. For the image data they use LeNet-5, which is a simple convolutional neural network introduced in 1989 by Yann LeCun for the task of recognizing handwritten digits [36]. The classical learner (Classic Rep.) uses the same encoder with a linear output layer. The results presented were found to be robust to small changes in training parameters such as learning rate or batch size. For more details on the training process we refer to Appendix A.4. All experiments required less than 3000 GPU-h to complete using NVIDIA Tesla T4 GPUs. In the next section we give brief descriptions of the data sets used in our experiments.

## 4.2 Data sets

In the following we first introduce two synthetic data sets designed specifically for this project before presenting three real world data sets from a broad range of domains.

**Square-Sign.** The synthetic data sets make use of a latent dynamical system to generate the latent variables $\{Z_t\}$. We choose a Gaussian distribution for $Z_1$, sample transition matrices $\{A_1, A_2, \ \dots \ A_T, \beta\}$ and compute the latent variables subsequently using the structural equations of Definition 3. For details on how the transition matrices and noise variables are sampled we refer to Appendix A.4. After computing the latent variables, we use a nonlinear deterministic injective observation function $\Psi$. For **Square-Sign** this nonlinear transformation $\Psi : \mathbb{R}^d \to \mathbb{R}^{2d}$ maps each latent feature $Z_{(t,k)}$ to a two dimensional vector such that

$$X_t \coloneqq \Psi(Z_{(t)}) = [Z^2_{(t,1)}, \operatorname{sgn}(Z_{(t,1)}), \dots, Z^2_{(t,d)}, \operatorname{sgn}(Z_{(t,d)})]^\top. \tag{4.1}$$

We point out that this nonlinear function meets the criteria laid out in Definition 3 and injective.

**Clocks-LGS.** The image data set **Clocks-LGS** is constructed very similarly to **Square-Sign**, as it makes use of the same latent-linear Gaussian system. The latent system has dimensionality $d = 2$, meaning $Z_t \in \mathbb{R}^2$. As observations this system generates grayscale images of 28 by 28 pixels which are reminiscent of clocks, hence the name. Each latent component is represented by one clock hand, one being a line pointing outwards from the center of the image which is marked with a circle, the other being a circle that orbits the image center. The exact value of each component is encoded in the position, size and fill of each of the two clock hands. The outcome is a linear function of the last latent state as in **Square-Sign**. We chose the image prediction task to test our algorithms on domains where neural networks are largely successful at finding good representations [37]. Further, we intend to investigate

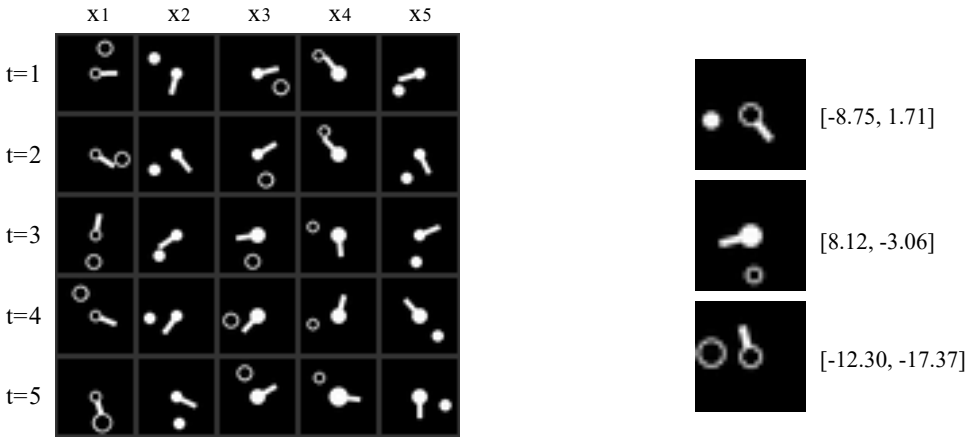



**Figure 4.1:** Example sequences of **Clocks-LGS** with $T = 5$ are displayed on the left. On the right we give three examples of how the latent states are represented as images. Both hands are at the top center for a value of $(0, 0)$ and arrive there again for $(2\pi, 2\pi)$ moving counterclockwise with increasing values.

whether the representation learning ideas presented in Section 3.4 are relevant for high dimensional observations such as images. For details on the image generation process please see Appendix A.4. Figure 4.1 displays five example sequences of **Clocks-LGS** on the left side and three observations $X_t$ with their corresponding latent variables $Z_t$ on the right.

**Alzheimer Progression ADNI** We also predict the progression of Alzheimer's disease of medical patients as measured by the outcome of the Mini Mental State Examination (MMSE) [38]. As part of the examination patients must answer a variety of questions and they are scored on their answers. The scores give an indication of the severity of the patient's condition. The anonymized data were obtained through the Alzheimer's Disease Neuroimaging Initiative (**ADNI**) [39] under the LONI research license. While patient measurements were taken every 3 months, the outcome of interest is the MMSE score 48 months after the first examination. Privileged information are the measurements taken at 12, 24 and 36 months. The **ADNI** data set was also used by Karlsson et al. with linear LuPTS, demonstrating that privileged time-series information is helpful in predicting the progression of Alzheimer. For more information on **ADNI** see Appendix A.4.

**Traffic.** The Metro Interstate traffic volume data set (**Traffic**) [40] contains hourly records of the westbound traffic volume on the interstate 94 highway between Minneapolis and St. Paul, MN as reported by the Minneapolis Department of Transport. In addition, the data contains weather features and a holiday indication. We predict the traffic volume for a fixed time horizon given the present observations. In our experiments, privileged information is observed every four hours. Figure 4.2 gives an example of how the traffic volume and the weather vary over the course of one month in this data set. For details on the preprocessing steps taken on this data set, we refer to Appendix A.4.

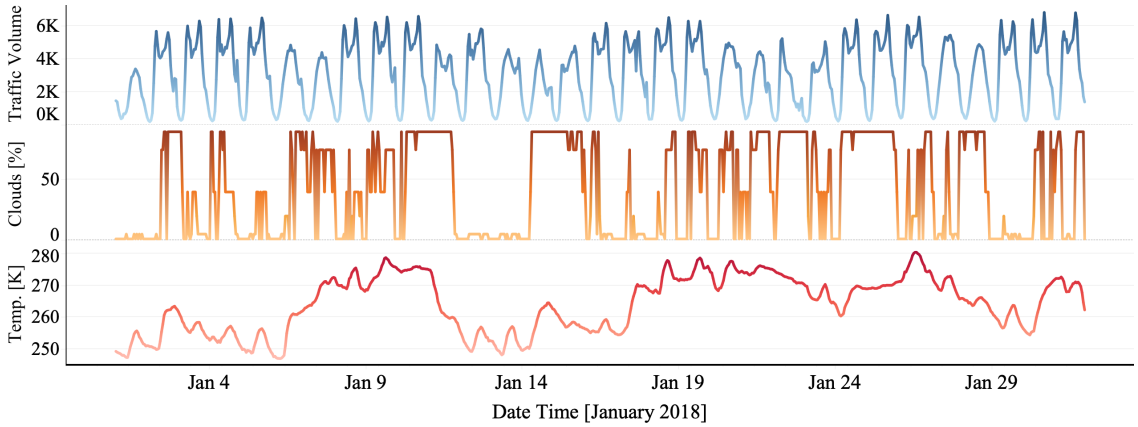
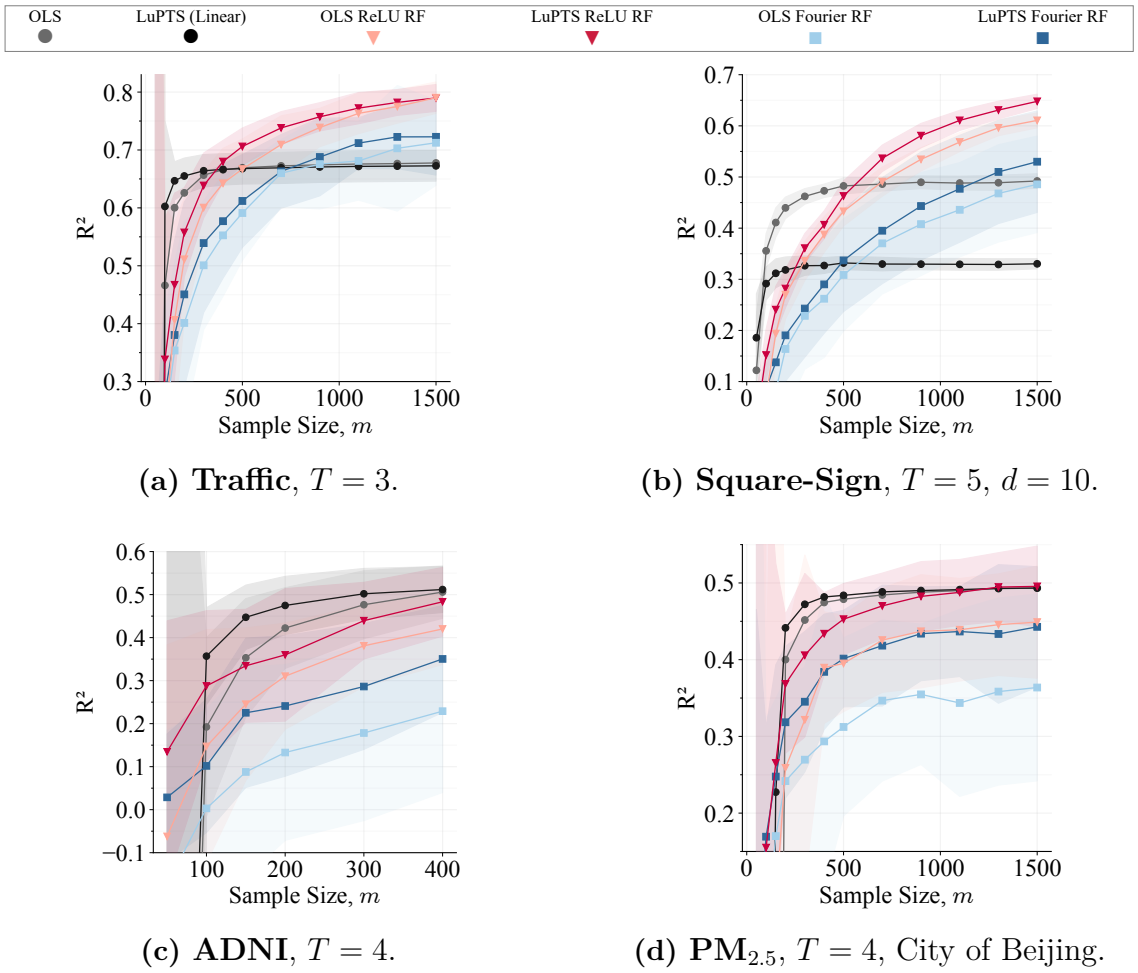



**Figure 4.2:** Traffic volume, cloud coverage in percent and the air temperature in Kelvin shown for the month of January 2018 as reported in the **Traffic** data set.

**$PM_{2.5}$ air quality.** At last we also predict the air quality in the five Chinese cities of Beijing, Guangzhou, Shanghai, Shenzen and Chengdu. To do so we use the **$PM_{2.5}$** data set as described by Liang et al. [41]. The data contains hourly measurements of the $PM_{2.5}$ particle concentration as well as weather features like temperature, air pressure, wind direction and others. Similarly to **ADNI** this data set was used in previous work with privileged time series by Hayashi et al.[11] and Karlsson et al [10]. For the exact preprocessing of the air quality data we refer to AppendixA.4.

## 4.3 Sample efficiency, bias & variance

The main goal of our work is to improve the sample efficiency of learning algorithms by incorporating privileged time-series information. We aim to make more accurate predictions using less data by using privileged information during the training process. Figure 4.3 shows the different variants (linear and nonlinear) of generalized LuPTS and the corresponding classical learners evaluated in terms of sample efficiency on all five data sets, excluding the image task. Across all prediction tasks and sample sizes, nonlinear variants of generalized LuPTS outperform their classical counterpart in terms of sample efficiency. On the **Traffic** prediction task, generalized LuPTS combined with random ReLU features [33] outperforms linear LuPTS as the former appears to exhibit less bias. On the synthetic data of **Square-Sign**, linear models reach their best accuracy quickly, while they are limited by their lack of expressiveness. Generalized LuPTS amplifies this bias, making linear LuPTS much worse than the classical learner (OLS) in this case. Random feature methods attain higher accuracy here eventually but are generally less sample efficient. Generalized LuPTS combined with random features manages to decrease this gap significantly, providing the benefits of nonlinear estimation with fewer samples. However, nonlinear models are not always superior to linear models even when the sample size increases: We do not see improvements using random features on the prediction tasks of **ADNI** and **$PM_{2.5}$** but the privileged learners are preferable to their classical counterpart there also.




**(a) Traffic**, $T = 3$.

**(b) Square-Sign**, $T = 5$, $d = 10$.

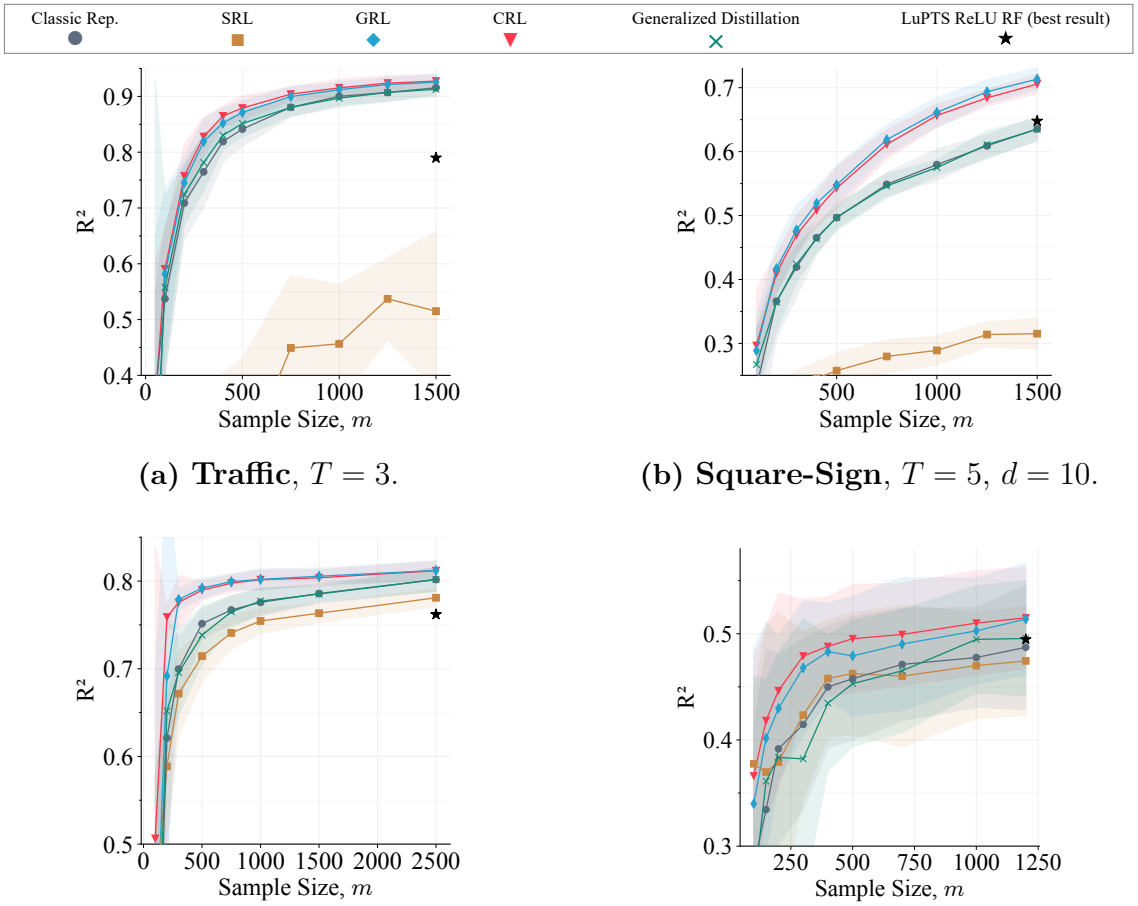
**(c) ADNI**, $T = 4$.

**(d) $PM_{2.5}$**, $T = 4$, City of Beijing.

**Figure 4.3:** Predictive accuracy of generalized LuPTS evaluated on four data sets, over 60 repetitions. The shaded area represents one standard deviation above and below the mean over repetitions.

The representation learners proposed in Section 3.4 are evaluated on the same data sets and on the image prediction task **Clocks-LGS**. In addition we include an approach that has been suggested by Hayashi et al [11]. They use generalized distillation (GD) as introduced by Lopez-Paz [3] for classification tasks and regard intermediate time points of time series as privileged information similar to the approach taken in this project. We adapt this setting to our regression problems to provide a baseline method. For the GD model we use the same architecture as for the classical learner and train it using a combined loss function with soft targets similar to (2.7), see Section 2.2. The soft targets are provided by a more capable teacher model that learns to predict the outcome from *all time steps* $X_1, X_2 \ \ldots \ X_T$ rather than just $X_1$. For details on the GD alternative we refer to Appendix A.4.

We present the predictive performance of the representation learners applied to four data sets in Figure 4.4. The results demonstrate that directly transferring the LuPTS objective to neural networks in the form of SRL results in subpar performance compared to the classic approach. When analyzing this phenomenon on more




**(a) Traffic**, $T = 3$.

**(b) Square-Sign**, $T = 5$, $d = 10$.

**(c) Clocks-LGS**, $T = 6$, $q = 1$.

**(d) $PM_{2.5}$**, $T = 4$, City of Beijing

**Figure 4.4:** Predictive accuracy of the representation learners over 25 repetitions. Generalized distillation is used as proposed by [11]. For details, see Appendix A.4.

prediction tasks with a variety of sequence lengths we noticed that this is particularly true for longer time series. When there is only a single step of privileged time-series information, SRL often performs better than classical learning. This is strongly reminiscent of the bias amplification effect discussed in Section 3.3. Overall, SRL does not seem to have a strong enough incentive to learn representations which accurately predict the outcome. Generalized distillation for privileged time series as suggested by Hayashi et al. [11], does not improve upon classical learning in our tasks, either.

On all experiments displayed in Figure 4.4, CRL and GRL outperform the classical learner. The predictive accuracy of these models is similar on most tasks, as may be explained by the fact that CRL may reduce to GRL when choosing $\lambda = 1$ in objective 3.17. Noticeably, the general observation of GRL and CRL being more sample efficient than the classic model, neither appears to depend on the neural architecture used for the encoder, nor does the modality of the data play an important role, as the image prediction task **Clocks-LGS** (Figure 4.4c) demonstrates. When comparing with these results with the best performance achieved by generalized LuPTS one notices that the representation learners are more sample efficient, despite requiring

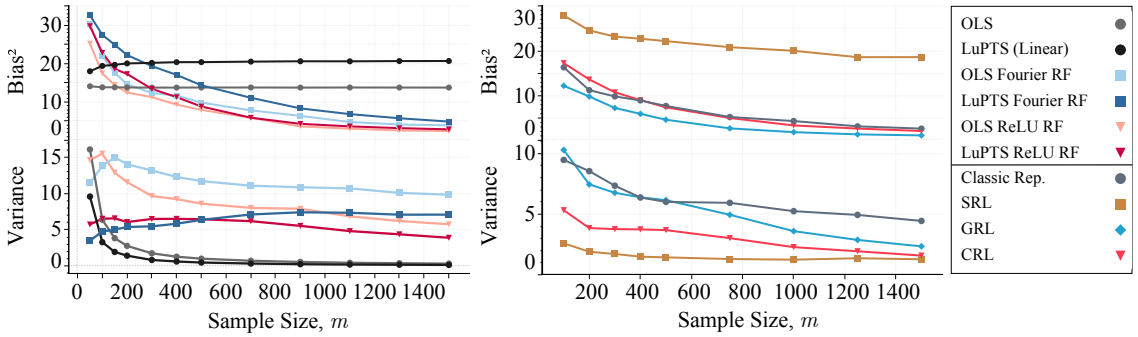



**Figure 4.5:** Estimated squared bias and variance of models trained on **Square-Sign** ($T = 5$, $d = 10$, $q = 3$) over random training sets (60 for left group, 25 for the right), evaluate on 1000 test points.

a validation set for training. For additional empirical results using other sequence lengths on the same data sets, we refer to Appendix A.6.

In order to understand how privileged information leads to more accurate predictions we analyze the bias and variance characteristics of our algorithms. As seen in Section 2.4, the prediction bias depends on the expected outcome given the baseline variables $\mathbb{E}[Y|X_1]$. This quantity is not accessible in real world data sets but on synthetic data, where the transition matrices and noise variables are known, we can estimate the expected squared prediction bias, $\mathbb{E}_{X_1}[(\mathbb{E}_D[\mathscr{A}(D)](X_1) - \mathbb{E}[Y|X_1])^2]$, by computing $\mathbb{E}_Y[Y|X_1]$. In addition we compute the variance of the different estimators analyze these metrics over a variety of training set sizes, while always predicting on the same 1000 samples of $X$ every time.

Figure 4.5 depicts bias and variance for all models on the **Square-Sign** data. On the left panel, all variants of generalized LuPTS exhibit lower variance than classical learning, despite being biased. This is remarkable as it is not at all guaranteed by Theorem 1. Generalized LuPTS only provably reduces variance when the representation function is known up to a linear transform and with random features methods this cannot be assumed. This is observed widely: *Across all experiments, we have never encountered an example where the use of privileged information has not resulted in lower variance compared to classical learning.* On the contrary, the privileged learners suffer higher bias than the comparable classical learners because of the bias compounding over the individual prediction steps as shown in Appendix A.3 and demonstrated empirically in Figure 3.4. For the random feature variants however, the bias decreases with the number of samples. This is obviously not the case for the linear models, their bias remains constant and here LuPTS attains higher bias than OLS.

The representation learners display similar characteristics on the right panel of Figure 4.5. First of all one notices that privileged time series learners offer a reduction in variance compared to classical learning (Classic Rep.) also here. Learning the transitions between latent variables $Z_t$ appears to be associated with low variance and high bias as demonstrated by the results of SRL, see objective (3.15). GRL

however, which does not model the single step forward transitions, exhibits the lowest bias and the largest variance of all privileged learners. As CRL is able to trade off between these two objectives, the estimates for its variance and bias lie in between the corresponding values of the other two privileged learners. While all these models use the same encoder combined with a linear model, one can view the additional loss terms used by the privileged learners as useful regularization using extra data which reduces variance and in some cases also bias (see GRL). The CRL algorithm can therefore be regarded as modelling the data generating process under the assumption of a latent dynamical system while also trading off between bias and variance.

Concluding, we point out that generalized LuPTS with random features benefits from privileged time-series information in all settings explored in this project and always appears to offer a reduction in variance compared to classical learning. While some settings do not benefit from nonlinear estimation others certainly do. In these settings generalized LuPTS used with random features requires less data to provide the advantages of nonlinear estimation compared to classical learning. Despite this gain, such methods may still exhibit high bias in small sample settings. As a remedy, we demonstrate representation learning based on neural network architectures that can trade off bias and variance more beneficially, further highlighting the great potential of learning with privileged time series. We witness a variance reduction when modelling intermediate steps of the sequences also here and make large improvements in prediction accuracy compared to classical learning in small sample settings.

## 4.4 Latent variable recovery

Under the assumptions made in Theorem 1, it is sufficient to identify the representation function $\Phi$ up to a linear transform $B$ to have provable gains from privileged information over a classical learner. This poses the question whether the representation learning algorithms of Section 3.4 find such a representation, whether privileged information is helpful in doing so and how this relates to their predictive performance. Moreover, representation learning is not only motivated by discovering representations that are useful for subsequent tasks such as making accurate predictions of the outcome but often it is also desired to recover the distinct causal signals that underlie the observed data. Nonlinear independent component analysis (nonlinear ICA) [42], which has also been applied to time series via contrastive learning, is a perfect example of this. This motivation paired with the knowledge that the latent variables $Z_t$ are accessible for synthetic data sets inspired us to assess to what extent a representation $\hat{\Phi}$ that is *linearly close* to the true representation has been found.

As a proxy for the existence of such a linear transform, we are interested in the mean correlation coefficients as produced by Canonical Correlation Analysis (CCA) [43]. Similar to principal component analysis (PCA) [44], CCA is a dimensional-

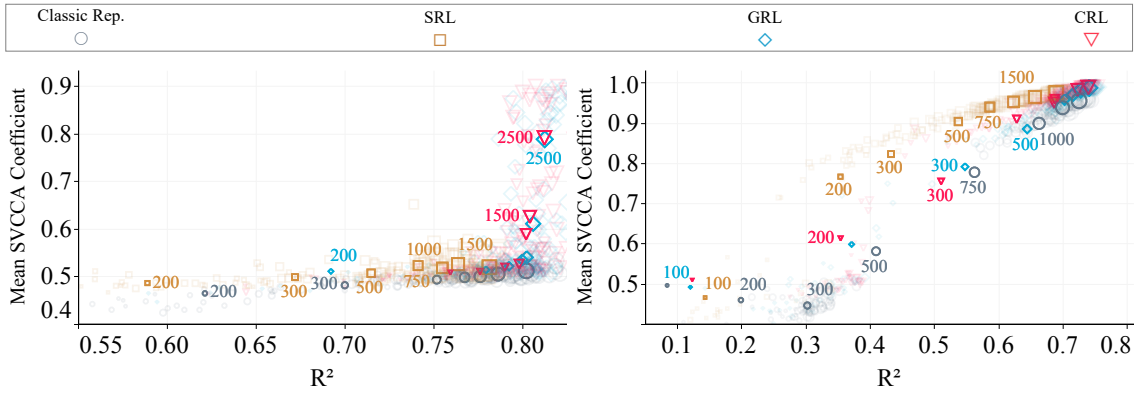



**Figure 4.6:** Mean $R^2$ and SVVCA coefficient over 25 training runs for the representation learning algorithms applied to **Clocks-LGS** with ($T = 5$,$q = 1$) on the left and ($T = 5$,$q = 2$) on the right. Faded markers show individual runs, solids markers represent averages over training runs for fixed sample size $m$. The annotations refer to the sample size of corresponding means.

ity reduction technique. Instead of retaining as much variance as possible with a given number of orthogonal components, CCA finds maximally correlated directions between two sets of data that can be of different dimensionality. Given design matrices of true representations $\mathbf{Z}_t$ and estimates of it $\hat{\mathbf{Z}}_t$, CCA sets out to find matrices $U \in \mathbb{R}^{d \times d}, V \in \mathbb{R}^{\hat{d} \times d}$ such that the correlation

$$\rho_j = Corr\Big((\mathbf{Z}_t U)_{(:,j)}, (\hat{\mathbf{Z}}_t V)_{(:,j)}\Big) \tag{4.2}$$

for each direction $j$ is maximized, where all directions are orthogonal to each other. One can then compute the correlation coefficients for all $j \in \{1, 2, \ldots, d\}$. We compute the mean $\bar{\rho}$ of the correlation coefficients over directions and time steps $t$, meaning $\bar{\rho}$ is equal to one when all $T$ estimated representations can be perfectly aligned with the true representations by CCA. In turn one gets $\bar{\rho} = 0$ when the representations are uncorrelated. As the latent states of neural networks contain noisy dimensions that result in erroneously high correlation coefficients when using CCA [45], we report the mean coefficients of Singular Vector Canonical Correlation Analysis (SVCCA) as presented by Raghu et al. [46]. This method first uses PCA [44] to remove noisy dimensions before applying CCA to the remaining components.

In Figure 4.6 we illustrate the mean SVCCA coefficients together with the prediction performance of the representation learners as measured by $R^2$. Higher is better in both metrics. We display the results of training runs for multiple sample sizes and repeat each training run 25 times for random data sets of the same size. On the left panel of Figure 4.6 the data generating process is configured for a one dimensional outcome $q = 1$, whereas the outcome is two dimensional on the right $q = 2$. The privileged learners GRL and CRL clearly do not only predict the outcome more accurately than the classic representation learner but also produce higher SVCCA coefficients when comparing the three for a fixed sample size. The difference is particularly large on the left panel of Figure 4.6 where the privileged learners need to recover two-dimensional latent variables to learn the dynamics accurately while only

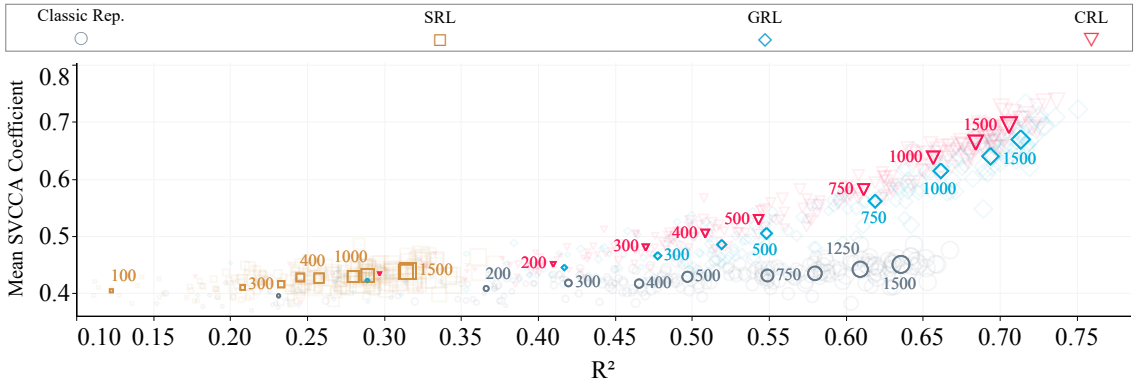



**Figure 4.7:** Coefficient of determination $R^2$ and mean SVCCA coefficients $\bar{\rho}$ for the different representation learning algorithms applied to the **Square-Sign** task with $(T = 5, d = 10, q = 3)$.

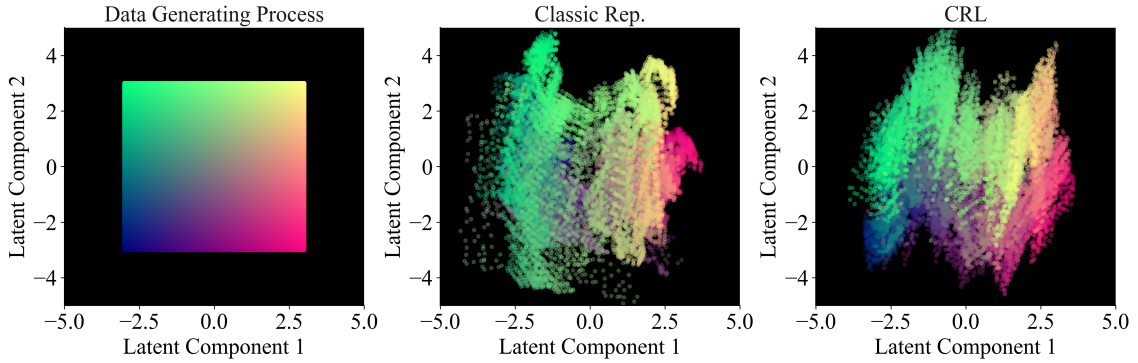



**Figure 4.8:** Visualizing the representations learned by Classic Rep. and CRL after applying SVCCA. The recovery target is shown on the left. Both estimators are best in class and were trained on 2500 samples.

a single latent dimension is useful in predicting the outcome. Similar observations can be made for the more simple problem of **Square-Sign** in Figure 4.7, where one can witness a strong trend of GRL and CRL outperforming the classical learner in terms of both metrics. Analyzing the results of SRL we note that it recovers the latent variables remarkably well when the latent dimensionality is equal to the outcome dimensionality $q = d = 2$ as seen on the right side of Figure 4.6.

To verify our findings with an example, we demonstrate the algorithm's capability to recover the latent variables using a visual example. To do so we visualize the representations learned by Classic Rep. and CRL on **Clocks-LGS** with a single outcome component $q = 1$ in Figure 4.8. Making use of the fact that we can access the true observation generating process $\Psi$ we construct a grid of $150 \times 150$ points on a square around the origin in the latent space $\mathcal{Z} \subseteq \mathbb{R}^2$. Every point is assigned a unique color on this grid as seen on the left of Figure 4.8. For each point $z$ we compute a clock image as an observation $x = \Psi(z)$ and pass it to the encoder of a representation learner, producing $\hat{z} = \hat{\Phi}(x)$. Just like above, we use SVCCA to reduce the dimensionality of the estimated representations $\hat{Z}$ and map them back

to the original latent space $\mathcal{Z}$ linearly. At last we plot the resulting representation with each point retaining its color. In the case of an optimal recovery, i.e. $\bar{\rho} = 1$, one would see a perfectly colored square as shown for the ground truth on the left. Analyzing the images, we notice that the representation produced by the classic representation learner has large gaps and that some colors are placed in the wrong regions (green in the left bottom corner). In contrast the right picture corresponding to CRL is much more coherent with the target. Both models displayed here produced the highest mean SVCCA coefficients for their model type and a sample size of $m = 2500$. Concluding, the use of privileged time-series information does not only increase the sample efficiency of existing algorithms but can also aid the recovery of latent variables in latent dynamical systems when combined with representation learning.

# 5
# Discussion

A large body of work is dedicated to improving the sample efficiency of learning algorithms with additional information. Existing analyses of learning using privileged information [2] [11], generalized distillation [3] or multi-view learning [47] [5] provide asymptotic learning rate improvements [7] or constant factor reduction of generalization bounds by reducing the size of the hypothesis class [8]. However, neither of these results claim that a privileged learner is provably preferable to a classical learner in small-sample settings as has been argued in the Introduction. In particular we are not aware of a theoretical result tight enough to establish a lower bound on the risk of a classical learner that is higher than the upper bound for a privileged learner.

Our problem is related to multi-task representation learning by Maurer et al. [48]. This is easily seen when reviewing the objective functions of GRL (3.16) and CRL (3.17). However, our problem is different in that only one task is of interest after learning. In estimation of causal effects, learning from *surrogate outcomes* [49] has been proposed to increase sample efficiency. Surrogate variables contain information about the outcome but are observed independently [50]. Just like privileged information surrogates may be viewed as side information to aid the learning process. However, this work assumes the surrogates might be observed when the outcome is unavailable. In other words surrogates are meant to compensate missing outcomes. Theoretical results do not give any guarantees in terms of sample efficiency when outcomes and surrogates are observed together at all times [51] [52].

Theorem 1 relies on the representation function $\Phi$ being identified up to a linear transformation. Nonlinear independent component analysis (ICA) aims to solve precisely this problem [53], which is disentangling the signals that have made up the observations using an unknown nonlinear function. It has been shown that non-stationary time series provide the conditions for identifiability through contrastive learning [54]. These results make for a potentially interesting connection between learning using privileged time series and nonlinear ICA. This is emphasized by the results of Section 4.4 where we demonstrate empirically that privileged time-series learning improves the recovery of latent variables.

As mentioned earlier, using privileged information that is observed as intermediate time points of time series has been studied by Hayashi et al. [11] and Karlsson et. al [10]. While the former uses logistic regression models to solve classification tasks, their results are not informative about regression tasks and do not provide

theoretical insights. The authors argue that taking a similar approach using neural networks might hold promise which has been confirmed empirically in Chapter 4. Karlsson et al. [10] proved the superiority of privileged learning in terms of sample efficiency in the case of linear-Gaussian dynamical data generating processes with linear estimators. Their work is certainly the foundation for what has followed as part of this project. Not only, did we extend their results to learning with kernels or feature maps in latent dynamical systems but we also demonstrated that the stepwise estimation approach of LuPTS is equivalent to classical learning in the underdetermined regime. We generalized their framework and can now argue about the benefits of privileged learners with much more general assumptions about the data generating process, however there might be cases when the assumption of a latent linear-Gaussian system is still too strong of a restriction, although we have not seen clear evidence of this in any of the prediction tasks considered.

There are a number of open questions that make for promising directions for future research in the field covered by this thesis. First of all, it would be desirable to make fewer structural assumptions about the data generating process and prove benefits of privileged time-series learners also then. One could imagine that a result similar to Theorem 1 could hold when the causal graph of the DGP is not a chain of linear transitions in the latent space but a directed acyclic graph with linear dependencies instead. Further, it is an open question whether privileged time series information is provably useful when the observation generating function is not injective, meaning one cannot hope to fully recover the true latent states. As generalized LuPTS amplifies bias and reduces variance compared to classical learning, it remains to be shown when exactly the variance reduction dominates this trade-off that occurs when the observation generating function is not known up to a linear transform like assumed by Theorem 1. Closely related to this question, we highlight that although we have never seen an example where the use of privileged information has resulted in increased variance, the variance reduction is not guaranteed by Theorem 1 in the case where the true representation function has not been recovered up to a linear transform. Proving that generalized LuPTS leads to a variance reduction even then would be an important milestone of future work. We showed that generalized LuPTS offers no advantages over classical learning when its kernel matrices are invertible as stated in Proposition 1. We avoided this condition by using random feature methods generating fewer features than samples but there might be other approaches worth investigating. Moreover, a regularized variant of the generalized LuPTS algorithm and corresponding theory would be desirable, as this could provide a privileged learning variant of kernel Ridge regression. We are optimistic that the utility of this work will inspire future research in these different directions so that the limitations mentioned will be overcome in order to provide a wide range of efficient learning algorithms that utilize privileged time-series information.

# 6

# Summary

The main goal considered by this project is to improve the sample efficiency of machine learning algorithms by incorporating privileged time-series information into the learning process. In particular, we set out to do this for the case where the outcome is likely to be a nonlinear function of the input. We approach this challenge by adapting the linear LuPTS algorithm of Karlsson et al. [10] to nonlinear settings through fixed representation functions or reproducing kernels which gives rise to the generalized LuPTS algorithm as introduced in Chapter 3.

We extend the existing framework of learning using privileged time series to latent dynamical systems and prove that privileged learning is preferable to classical learning when the function connecting latent variables and observations is known up to a linear transform. In doing so we demonstrate that generalized LuPTS undergoes a phase transition around the point where the number of samples is equal to the number of features, after which the gains from privileged information vanish in the underdetermined regime.

For the case when representation functions are unknown we combine generalized LuPTS with random feature methods such as random Fourier features [32] or random ReLU features [33] and show that this can be turned into a consistent learning algorithm. We empirically demonstrate a variance reduction through the use of privileged information using these methods despite the fact that this is not always guaranteed by existing theory. It is pointed out that generalized LuPTS with random features makes more accurate predictions than classical learning in settings where linear LuPTS fails dramatically due to high bias. However, we also show that combining privileged time series learning with random features might still produce large bias in small sample settings. As a remedy, we propose representation learning algorithms using privileged information to trade off bias and variance more efficiently.

We verify our theoretical findings for generalized LuPTS empirically on two synthetic and three real world data sets. We analyze the bias and variance characteristics for all algorithms and witness a reduction in variance for all privileged learners when comparing to the classical alternative. For the representation learning algorithms the empirical results show even greater performance compared to generalized LuPTS when using privileged information. Analyzing the ability of these models to recover the latent variables of latent dynamical systems, we see advantages of using privileged information also there.

We discuss how our work relates to existing results on similar problems and point out that we are not aware of other results that establish a clear preference for privileged learners on small sample settings with nonlinear estimation. At last, exciting directions for future research based on the findings of this project are identified, as the benefits of learning with privileged information will hopefully prove to be valuable in real-world applications with small sample sizes, in the near future.

# Bibliography


[1] C. Lee and H.-J. Yoon, "Medical big data: Promise and challenges," *Kidney Research and Clinical Practice*, vol. 36, pp. 3–11, Mar. 2017. DOI: 10.23876/j.krcp.2017.36.1.3.

[2] V. Vapnik and A. Vashist, "A new learning paradigm: Learning using privileged information," *Neural Networks*, vol. 22, no. 5, pp. 544–557, 2009, Advances in Neural Networks Research: IJCNN2009, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2009.06.042. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608009001130.

[3] D. Lopez-Paz, L. Bottou, B. Schölkopf, and V. Vapnik, "Unifying distillation and privileged information," *arXiv preprint arXiv:1511.03643*, 2015.

[4] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, 2015. arXiv: 1503.02531 [stat.ML].

[5] D. S. Rosenberg and P. L. Bartlett, "The rademacher complexity of co-regularized kernel classes," in *Artificial Intelligence and Statistics*, PMLR, 2007, pp. 396–403.

[6] V. Vapnik, R. Izmailov, *et al.*, "Learning using privileged information: Similarity control and knowledge transfer.," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 2023–2049, 2015.

[7] D. Pechyony and V. Vapnik, "On the theory of learnining with privileged information," *Advances in neural information processing systems*, vol. 23, 2010.

[8] W. Wang, "Everything old is new again: A multi-view learning approach to learning using privileged information and distillation," *arXiv preprint arXiv:1903.03694*, 2019.

[9] R. Jonschkowski, S. Höfer, and O. Brock, "Patterns for learning with side information," *arXiv preprint arXiv:1511.06429*, 2015.

[10] R. Karlsson, M. Willbo, Z. Hussain, R. G. Krishnan, D. Sontag, and F. D. Johansson, *Using time-series privileged information for provably efficient learning of prediction models*, 2021. arXiv: 2110.14993 [cs.LG].

[11] S. Hayashi, A. Tanimoto, and H. Kashima, "Long-term prediction of small time-series data using generalized distillation," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851687.

[12] R. Penrose, "On best approximate solutions of linear matrix equations," in *Mathematical Proceedings of the Cambridge Philosophical Society*, Cambridge University Press, vol. 52, 1956, pp. 17–19.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, ISBN: 0387310738.

[14] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 42, no. 1, pp. 80–86, 2000, ISSN: 00401706. [Online]. Available: http://www.jstor.org/stable/1271436 (visited on 05/22/2022).

[15] B. Schölkopf, A. J. Smola, and F. Bach, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2018, ISBN: 0262536579.

[16] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *Computational Learning Theory*, D. Helmbold and B. Williamson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 416–426, ISBN: 978-3-540-44581-4.

[17] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[18] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019. DOI: 10.1561/2200000056. [Online]. Available: https://doi.org/10.1561%2F2200000056.

[19] Z. Liu, Y. Lin, and M. Sun, *Representation learning for natural language processing*. Springer Nature, 2020.

[20] F. Chen, Y.-C. Wang, B. Wang, and C.-C. J. Kuo, "Graph representation learning: A survey," *APSIPA Transactions on Signal and Information Processing*, vol. 9, e15, 2020. DOI: 10.1017/ATSIP.2020.13.

[21] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[22] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.* 1958. DOI: 10.1037/h0042519. [Online]. Available: http://dx.doi.org/10.1037/h0042519.

[23] B. Mehlig, *Machine Learning with Neural Networks*. Cambridge University Press, Oct. 2021. DOI: 10.1017/9781108860604. [Online]. Available: https://doi.org/10.1017%2F9781108860604.

[24] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 1836–1841. DOI: 10.1109/CCDC.2018.8407425.

[25] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[26] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information*

*Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[27] K. Lee and K. Carlberg, *Deep conservation: A latent-dynamics model for exact satisfaction of physical conservation laws*, 2019. DOI: 10.48550/ARXIV.1909.09754.

[28] J. C. Kao, P. Nuyujukian, S. I. Ryu, M. M. Churchland, J. P. Cunningham, and K. V. Shenoy, "Single-trial dynamics of motor cortex and their applications to brain-machine interfaces," *Nature Communications*, vol. 6, no. 1, Jul. 2015. DOI: 10.1038/ncomms8759.

[29] M. Loog, T. Viering, A. Mey, J. H. Krijthe, and D. M. J. Tax, "A brief prehistory of double descent," *Proceedings of the National Academy of Sciences*, vol. 117, no. 20, pp. 10 625–10 626, 2020. DOI: 10.1073/pnas.2001875117. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2001875117.

[30] A. Caponnetto, C. A. Micchelli, M. Pontil, and Y. Ying, "Universal multi-task kernels," *The Journal of Machine Learning Research*, vol. 9, pp. 1615–1646, 2008.

[31] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, Jun. 2008.

[32] A. Rahimi, B. Recht, *et al.*, "Random features for large-scale kernel machines.," in *NIPS*, Citeseer, vol. 3, 2007, p. 5.

[33] Y. Sun, A. Gilbert, and A. Tewari, *On the approximation properties of random relu features*, 2018. DOI: 10.48550/ARXIV.1810.04374.

[34] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Rob. Res.*, vol. 32, no. 11, pp. 1238–1274, Sep. 2013, ISSN: 0278-3649. DOI: 10.1177/0278364913495721.

[35] J. Zhao, X. Xie, X. Xu, and S. Sun, "Multi-view learning overview: Recent progress and new challenges," *Information Fusion*, vol. 38, pp. 43–54, 2017, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2017.02.007.

[36] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. eprint: https://direct.mit.edu/neco/article-pdf/1/4/541/811941/neco.1989.1.4.541.pdf.

[37] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013. DOI: 10.1109/TPAMI.2013.50.

[38] M. Galea and M. Woodward, "Mini-mental state examination (mmse)," *Australian Journal of Physiotherapy*, vol. 51, no. 3, p. 198, 2005, ISSN: 0004-9514. DOI: https://doi.org/10.1016/S0004-9514(05)70034-9.

[39] ADNI, *The Alzheimer's Disease Neuroimaging Initiative (ADNI)*, 2004. [Online]. Available: http://adni.loni.usc.edu.

[40] J. Hogue, *Metro interstate traffic volume data set*, 2012. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume.

[41] X. Liang, S. Li, S. Zhang, H. Huang, and S. X. Chen, "Pm2.5 data reliability, consistency, and air quality assessment in five chinese cities," *Journal of Geophysical Research*, vol. 121, 17 2016, ISSN: 21562202. DOI: 10.1002/2016JD024877.

[42] R. P. Monti, K. Zhang, and A. Hyvärinen, "Causal discovery with general non-linear relationships using non-linear ica," in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, R. P. Adams and V. Gogate, Eds., ser. Proceedings of Machine Learning Research, vol. 115, PMLR, Jul. 2020, pp. 186–195. [Online]. Available: https://proceedings.mlr.press/v115/monti20a.html.

[43] H. Hotelling, "Relations between two Sets of Variates*," *Biometrika*, vol. 28, no. 3-4, pp. 321–377, Dec. 1936, ISSN: 0006-3444. DOI: 10.1093/biomet/28.3-4.321. eprint: https://academic.oup.com/biomet/article-pdf/28/3-4/321/586830/28-3-4-321.pdf.

[44] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.

[45] A. S. Morcos, M. Raghu, and S. Bengio, *Insights on representational similarity in neural networks with canonical correlation*, 2018. DOI: 10.48550/ARXIV.1806.05759. [Online]. Available: https://arxiv.org/abs/1806.05759.

[46] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, *Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability*, 2017. DOI: 10.48550/ARXIV.1706.05806.

[47] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 92–100.

[48] A. Maurer, M. Pontil, and B. Romera-Paredes, "The benefit of multitask representation learning," *Journal of Machine Learning Research*, vol. 17, no. 81, pp. 1–32, 2016.

[49] R. L. Prentice, "Surrogate endpoints in clinical trials: Definition and operational criteria," *Statistics in medicine*, vol. 8, no. 4, pp. 431–440, 1989.

[50] S. Athey, R. Chetty, G. W. Imbens, and H. Kang, "The surrogate index: Combining short-term proxies to estimate long-term treatment effects more rapidly and precisely," National Bureau of Economic Research, Tech. Rep., 2019.

[51] S. X. Chen, D. H. Leung, and J. Qin, "Improving semiparametric estimation by using surrogate data," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 4, pp. 803–823, 2008.

[52] N. Kallus and X. Mao, "On the role of surrogates in the efficient estimation of treatment effects with limited outcome data," *arXiv preprint arXiv:2003.12408*, 2020.

[53] A. Hyvärinen and P. Pajunen, "Nonlinear independent component analysis: Existence and uniqueness results," *Neural networks*, vol. 12, no. 3, pp. 429–439, 1999.

[54] A. Hyvarinen and H. Morioka, "Unsupervised feature extraction by time-contrastive learning and nonlinear ica," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[56] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[57] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[58] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[59] X. Liang, T. Zou, B. Guo, *et al.*, "Assessing beijing's pm2.5 pollution: Severity, weather impact, apec and winter heating," English, *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 471, no. 2182, Oct. 2015, ISSN: 1364-5021. DOI: 10.1098/rspa.2015.0257.

# A

# Appendix

## A.1 Proof of Theorem 1

Our proof requires an additional technical assumption: that the matrix of true latent states $\mathbf{Z}_t = [\Phi(x_{1,t}), ..., \Phi(x_{m,t})]^\top$, for all $t$, for a random data set $D$ has independent columns with probability 1. This implies that $\text{rank}(\mathbf{Z}_t) = d$ and $m \geq d$. We consider this a minor restriction since it would only be violated if either a) two or more components of $Z_t$ were perfectly correlated—in this case, a smaller system with same distributions over observations could always be constructed—or b) if we observe fewer samples than necesary to determine the system ($m < d$). Note that this *does not* require that the dimension $\hat{d}$ of the estimated representation $\hat{\Phi}$ is smaller than $m$. We begin by proving that both classical and LuPTS estimators are invariant to a particular form of linear transformation of the representation $\hat{\Phi}$.

**Lemma 1.** *Assume we have a latent linear Gaussian system as desribed in Definition 3 such that for a data set $D$ of $m$ samples, the matrix of true latent states $\mathbf{Z}_t = [\Phi(X_{1,t}); ...; \Phi(X_{m,t})]$ has linearly independent columns with probability 1. Let $\mathscr{A}_P^\Phi$ be the LuPTS algorithm using the system's true map $\Phi(\cdot)$ with $\Phi(\Psi(z)) = z\ \forall z \in \mathcal{Z}$. Let $\mathscr{A}_P^{\hat{\Phi}}$ be the same algorithm using a different map $\hat{\Phi}(\cdot)$. We assume that $\exists B : \hat{\Phi}(x) = B\Phi(x) \forall x \in \mathcal{X}$. Analogously, we denote the classical learners $\mathscr{A}_C^{\hat{\Phi}}$ and $\mathscr{A}_C^\Phi$. If $B \in \mathbb{R}^{\hat{d}\times d}$ has linearly independent columns we have*

$$\hat{h}_P^{\hat{\Phi}}(x) = \hat{h}_P^{\Phi}(x)$$
$$\textit{and}\ \ \hat{h}_C^{\hat{\Phi}}(x) = \hat{h}_C^{\Phi}(x).$$

*Proof.* Let $\mathbf{Z}_t \in \mathbb{R}^{m\times d}$ be made up of the rows $\mathbf{Z}_{t(i,:)} = \Phi(\mathbf{X}_{t(i,:)})$ when $\mathbf{X}_t \in \mathbb{R}^{m\times k}$ is the design matrix belonging to data set $D$. In the same fashion we define $\hat{\mathbf{Z}}_t \in \mathbb{R}^{m\times\hat{d}}$ using the map $\hat{\Phi}$ instead. By assumption, $\mathbf{Z}_t$ and $B \in \mathbb{R}^{\hat{d}\times d}$ have independent columns such that $B^\dagger B = I$. These assumptions are used for matrix identities involving the Moore-Penrose inverse below. We compute the prediction on a new

test point $x$ for the classical learner:

$$\begin{aligned}
\hat{h}_{\mathrm{C}}^{\hat{\Phi}}(x) &= \left((\hat{\mathbf{Z}}_1^\top \hat{\mathbf{Z}}_1)^\dagger \hat{\mathbf{Z}}_1^\top Y\right)^\top \hat{\Phi}(x) \\
&= \left(\left((\mathbf{Z}_1 B^\top)^\top (\mathbf{Z}_1 B^\top)\right)^\dagger (\mathbf{Z}_1 B^\top)^\top \mathbf{Y}\right)^\top B\Phi(x) \\
&= \left((\mathbf{Z}_1 B^\top)^\dagger (B\mathbf{Z}_1^\top)^\dagger (\mathbf{Z}_1 B^\top)^\top \mathbf{Y}\right)^\top B\Phi(x) \\
&= \left((B^\top)^\dagger \mathbf{Z}_1^\dagger (\mathbf{Z}_1^\top)^\dagger B^\dagger (\mathbf{Z}_1 B^\top)^\top \mathbf{Y}\right)^\top B\Phi(x) \\
&= \left((B^\top)^\dagger \left(\mathbf{Z}_1^\top \mathbf{Z}_1\right)^\dagger \underbrace{B^\dagger B}_{=I} \mathbf{Z}_1^\top \mathbf{Y}\right)^\top B\Phi(x) \\
&= \left(\left(\mathbf{Z}_1^\top \mathbf{Z}_1\right)^\dagger \mathbf{Z}_1^\top \mathbf{Y}\right)^\top \underbrace{B^\dagger B}_{=I} \Phi(x) \\
&= \hat{h}_{\mathrm{C}}^{\Phi}(x)
\end{aligned}$$

The arguments for the privileged learners are analogous:

$$\begin{aligned}
\hat{h}_{\mathrm{P}}^{\hat{\Phi}}(x) &= \Big[(\hat{\mathbf{Z}}_1^\top \hat{\mathbf{Z}}_1)^\dagger \hat{\mathbf{Z}}_1^\top \hat{\mathbf{Z}}_2 \ (\hat{\mathbf{Z}}_2^\top \hat{\mathbf{Z}}_2)^\dagger \hat{\mathbf{Z}}_2^\top \hat{\mathbf{Z}}_3 \ \dots \ (\hat{\mathbf{Z}}_T^\top \hat{\mathbf{Z}}_T)^\dagger \hat{\mathbf{Z}}_T^\top \mathbf{Y}\Big]^\top \hat{\Phi}(x) \\
&= \Big[(B^\top)^\dagger (\mathbf{Z}_1^\top \mathbf{Z}_1)^\dagger B^\dagger B \mathbf{Z}_1^\top \mathbf{Z}_2 B^\top (B^\top)^\dagger (\mathbf{Z}_2^\top \mathbf{Z}_2)^\dagger B^\dagger B \mathbf{Z}_2^\top \mathbf{Z}_3 B^\top \\
&\quad \dots \ (B^\top)^\dagger (\mathbf{Z}_T^\top \mathbf{Z}_T)^\dagger B^\dagger B \mathbf{Z}_T^\top \mathbf{Y}\Big]^\top B\Phi(x) \\
&= \Big[(\mathbf{Z}_1^\top \mathbf{Z}_1)^\dagger \mathbf{Z}_1^\top \mathbf{Z}_2 (\mathbf{Z}_2^\top \mathbf{Z}_2)^\dagger \mathbf{Z}_2^\top \mathbf{Z}_3 (\mathbf{Z}_T^\top \mathbf{Z}_T)^\dagger \mathbf{Z}_T^\top \mathbf{Y}\Big]^\top B^\dagger B\Phi(x) \\
&= \hat{h}_{\mathrm{P}}^{\Phi}(x)
\end{aligned}$$

□

**Proof of Theorem 1.** We consider the generalized LuPTS estimator $\hat{h}_{\mathrm{P}}^{\hat{\Phi}}(\cdot)$ treating different cases for the number of samples $m$ and latent state dimension $d$ in turn. By the added technical assumption, that the true latent state $\mathbf{Z}_t = [\Phi(x_{1,t}), ..., \Phi(x_{m,t})]^\top$ has rank $d$ for all $t$ with probability 1, and the assumption that $\hat{\Phi}$ is linearly close to $\Phi$, by a matrix $B \in \mathbb{R}^{\hat{d}\times d}$ of rank $d$ such that $\hat{\mathbf{Z}}_t = \mathbf{Z}_t B^\top$, we get that $\mathrm{rank}(\hat{\mathbf{Z}}_t) = d$ for all $t$. This also implies $d \leq \hat{d}$.

(i) $m = d$: In this case, the Gram matrix $\mathbf{K}_t = \hat{\mathbf{Z}}_t \hat{\mathbf{Z}}_t^\top$ has full rank and thus is invertible for all $t$. By Proposition 1, LuPTS coincides with the classical learner. Hence, $\mathbb{E}_{\hat{h}_{\mathrm{P}}, X_1}[\mathrm{Var}_D(\hat{h}_{\mathrm{C}}(X_1) \mid \hat{h}_{\mathrm{P}})] = 0$ and $\overline{R}(\mathscr{A}_{\mathrm{P}}) = \overline{R}(\mathscr{A}_{\mathrm{C}})$.

(ii) $m < d$: In this case, there does not exists a linearly close, as defined above, representation $\hat{\Phi}$ to $\Phi$ since the rank of $\hat{\mathbf{Z}}_t$ must be smaller than $d$. This contradicts

that $\operatorname{rank}(\hat{\mathbf{Z}}_t) = d$. Independently, if the conditions of Proposition 1 hold, the same equivalence holds as in the case $m = d$.

(iii) $m > d$: In this case, the kernel Gram matrix $\mathbf{K}_t = \hat{\mathbf{Z}}_t \hat{\mathbf{Z}}_t^\top$ has rank $d < m$ and is never invertible.

Three sub-cases remain: a) When $\hat{d} = d$, the matrix $B$ is invertible and square, the covariance matrix $\hat{\Sigma}_t = \hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_t$ is invertible for all $t$ and our estimator coincides with linear LuPTS [10] in the space implied by $\hat{\Phi}$. To see this, note that Lemma 1 implies that $\hat{h}_{\text{P}}^{\hat{\Phi}}(\cdot)$ makes the same predictions as a different generalized LuPTS estimator $\hat{h}_{\text{P}}^{\Phi}(\cdot)$ using the true map $\Phi$ when the two representation functions are related through $B$, as defined in the Theorem statement. Consequently, we may analyze the latter estimator instead of the first. It uses the parameter

$$\hat{\theta}_{\text{P}} \coloneqq \left[ \prod_{t=1}^{T-1} \underbrace{(\mathbf{Z}_t^\top \mathbf{Z}_t)^\dagger \mathbf{Z}_t^\top \mathbf{Z}_{t+1}}_{\hat{A}_t} \right] \underbrace{(\mathbf{Z}_T^\top \mathbf{Z}_T)^\dagger \mathbf{Z}_T^\top \mathbf{Y}}_{\hat{\beta}} \ .$$

We know by assumption that the covariance matrices $\Sigma_t = (\mathbf{Z}_t^\top \mathbf{Z}_t) \in \mathbb{R}^{d \times d}$ have full rank for all $t$. This implies that the Moore-Penrose pseudoinverse $(\cdot)^\dagger$ may be replaced by the regular matrix inverse $(\cdot)^{-1}$ in the expression above, yielding

$$\hat{\theta}_{\text{P}} = \left[ \prod_{t=1}^{T-1} \underbrace{(\mathbf{Z}_t^\top \mathbf{Z}_t)^{-1} \mathbf{Z}_t^\top \mathbf{Z}_{t+1}}_{\hat{A}_t} \right] \underbrace{(\mathbf{Z}_T^\top \mathbf{Z}_T)^{-1} \mathbf{Z}_T^\top \mathbf{Y}}_{\hat{\beta}}$$

which is equivalent to the LuPTS estimator of [10] used on a linear-Gaussian system in the space $\mathcal{Z}$ rather than in $\mathcal{X}$. In this case, Theorem 1 from Karlsson et al.[10] yields the desired result. b) If $\hat{d} > d$, $\hat{\Sigma}$ is not invertible but, due to Lemma 1, we can instead study a representation which is an appropriate linear transform $B \in \mathbb{R}^{\hat{d} \times d}$ away from $\hat{\mathbf{Z}}$, and apply the result of Karlsson et al.[10] as described for the case $\hat{d} = d$. Note that in this case $B$ is non-square but has linearly independent columns as required. c) If $\hat{d} < d$, the assumed matrix $B$ cannot exist with the stated conditions (the assumptions of Theorem 1 are not satisfied).

□

## A.2 Universality of random features for LuPTS

A learning algorithm $\mathscr{A}$ is said to be universally consistent if, for any continuous function $h$, the output of $\mathscr{A}$ converges in probability to $h$. That is, for a random dataset $D_m$ of $m$ i.i.d. samples drawn from a distribution $p$, and any $\epsilon > 0$,

$$\lim_{m \to \infty} \Pr[\|\mathscr{A}(D_m) - h\|_{L^2(p)} > \epsilon] = 0 \ .$$

Sun et al. [33] prove that (norm-bounded) linear regression applied to random ReLU features (RRF) is universally consistent. Specifically, the estimator

$$\hat{h}_{\text{RRF}}(x) = \hat{\theta}^\top \hat{\Phi}_{\text{RRF}}^{\gamma,\hat{d}}(x) \quad \text{with} \quad \hat{\theta} = \underset{\theta : \|\theta\|_2^2 < R^2}{\arg\min} \frac{1}{m} \sum_{i=1}^{m} (\theta^\top \hat{\Phi}_{\text{RRF}}^{\gamma,\hat{d}}(x_i) - y_i)^2$$

is universally consistent for univariate continuous functions of $x$. We can apply the same result for LuPTS by considering each parameter estimate of the latent dynamical system given $\hat{\Phi}$, with norms restricted by $R$,

$$[\hat{A}_t]_{:,j} = \underset{a:\|a\|_2^2<R^2}{\arg\min} \frac{1}{m}\sum_{i=1}^{m}(a^\top \hat{\Phi}_{\mathrm{RRF}}^{\gamma,\hat{d}}(x_{i,t}) - \hat{\Phi}_{\mathrm{RRF}}^{\gamma,\hat{d}}(x_{i,t+1})_j)^2 \quad \text{for} \quad j \in [\hat{d}], t \in [T-1] \tag{A.1}$$

$$\hat{\beta} = \underset{b:\|b\|_2^2<R^2}{\arg\min} \frac{1}{m}\sum_{i=1}^{m}(b^\top \hat{\Phi}_{\mathrm{RRF}}^{\gamma,\hat{d}}(x_{i,T}) - y_i)^2 \tag{A.2}$$

Let $\mathcal{H}_{\mathrm{ReLU}}$ denote the set of linear functions applied to $\hat{d}$ random ReLU features, with uniform random projection coefficients $\omega_j \sim \mathcal{U}(\mathbb{S}^d)$, $j = 1, ..., \hat{d}$.

$$\mathcal{H}_{\mathrm{ReLU},\hat{d}} = \left\{ h : \mathcal{X} \to \mathbb{R}, h(x) = \sum_{j=1}^{\hat{d}} a_j \sigma(\omega_j^\top [x; 1]) \right\} .$$

**Corollary 1** (Follows from Proposition 5 in [33])**.** *Let data be produced by latent-dynamical system just as described in Definition 3 with noiseless transitions and outcomes, $U_t = 0$ for $t = 2, ..., T$, and $U_Y = 0$. Define $g_t^*(x_t) \coloneqq \Psi(A_t^\top \Phi(x_t)) = x_{t+1}$ and assume that for any fixed RRF representation $\hat{\Phi}_{\hat{d}}$, each component $j = 1, ..., \hat{d}$ of the transition target satisfies $\hat{\Phi}_{\hat{d}}(g_t^*(\cdot))(j) \in \mathcal{H}_{\mathrm{ReLU},\hat{d}}$. Let $\hat{A}_t$ be the minimizer of the single-step transitions as defined in* (A.1)*. Then, for any $\epsilon > 0$ and a large enough number of random features $\hat{d}$ and samples $m$, with probability $\geq 1 - \delta$,*

$$\|\hat{A}_t^\top \hat{\Phi}_{\hat{d}}(x_t) - \hat{\Phi}_{\hat{d}}(x_{t+1})\| \leq \epsilon .$$

The result is a special case of Proposition 5 in [33] applied to the transition functions in our problem. Putting this together for all time-steps, we get the following result.

**Proposition 2** (Universal consistency of RRF LuPTS)**.** *Assume that $U \equiv 0, U_Y \equiv 0$. By the consistency of RRF regression, we have for a sufficiently large number of random features $\hat{d}$ and samples $m$, that with probability at least $1 - \delta/T$, for estimates* (A.1)*,* (A.2)

$$\|\hat{\beta}^\top \hat{\Phi}(X_T) - Y\|_{L^2(p)} \leq \epsilon$$

*and*

$$\forall t = 1, ..., T-1 : \|\hat{A}_t^\top \hat{\Phi}(X_t) - \hat{\Phi}(X_{t+1})\|_{L^2(p)} \leq \epsilon,$$

*Then, further assume that the largest eigenvalue $\lambda_{\max}(\hat{A}_t) \leq R$ for any $t$ and $\|\hat{\beta}\| \leq R$. Then, with probability at least $1 - \delta$,*

$$\|(\hat{A}_1 \cdots \hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_1) - Y\|_{L^2(p)} \leq \mathcal{O}(TR^T\epsilon)$$

*Proof.* Let $\|\cdot\| = \|\cdot\|_{L^2(p)}$. Then, applying a union bound to each of the $T$ $(\epsilon, \delta)$-

assumptions

$$
\begin{aligned}
&\|(\hat{A}_1 \cdots \hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_1) - Y\| = \|(\hat{A}_1 \cdots \hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_1) - Y\| \\
&= \|(\hat{A}_1 \cdots \hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_1) - \hat{\beta}^\top \hat{\Phi}(X_T) + \hat{\beta}^\top \hat{\Phi}(X_T) - Y\| \\
&\leq \|(\hat{A}_1 \cdots \hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_1) - \hat{\beta}^\top \hat{\Phi}(X_T)\| + \underbrace{\|\hat{\beta}^\top \hat{\Phi}(X_T) - Y\|}_{\leq \epsilon} \\
&\leq \|(\hat{A}_1 \cdots \hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_1) - \hat{\beta}^\top \hat{A}_{T-1}^\top \hat{\Phi}(X_{T-1}) + \hat{\beta}^\top (\hat{A}_{T-1}^\top \hat{\Phi}(X_{T-1}) - \hat{\Phi}(X_T))\| + \epsilon \\
&\leq \|(\hat{A}_1 \cdots \hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_1) - (\hat{A}_{T-1}\hat{\beta})^\top \hat{\Phi}(X_{T-1})\| + \underbrace{\|\hat{\beta}^\top (\hat{A}_{T-1}^\top \hat{\Phi}(X_{T-1}) - \hat{\Phi}(X_T))\|}_{\leq R\epsilon} + \epsilon \\
&\ldots \\
&\leq TR^T\epsilon \ .
\end{aligned}
$$

□

Proposition 2 shows that LuPTS with fandom ReLU features can be turned into a universally consistent estimator of any (noiseless) continuous function of $X_1$ by adding a norm constraint to each linear parameter.

## A.3 Compounding bias

We can describe the compounding bias of the LuPTS estimator due to a biased representation $\hat{\Phi}$, in comparison with the standard OLS estimator, by propagating the error in $\hat{\Phi}$ through the estimates. Assume that

$$Y = \theta^\top \Phi(X_1) + \epsilon = (A_1 \cdots A_{T-1}\beta)^\top \Phi(X_1) + \epsilon'$$

Then, let $Z_t = \Phi(X_t)$ and for an estimate $\hat{\Phi}$, assumed for simplicity to have the same dimension, $\hat{d} = d$,

$$\hat{Z}_t = \hat{\Phi}(X_t) = \Phi(X_t) + R_t$$

where $R_t$ is the residual w.r.t. $\Phi$. Let bold-face variables indicate multi-sample equivalents of all variables. Further, define $\Sigma_t = \mathbf{Z}_t^\top \mathbf{Z}_t$ and $\hat{\Sigma}_t = \hat{\mathbf{Z}}_t^\top \hat{\mathbf{Z}}_t$.
Fitting $\hat{\theta}$ to $\hat{\Phi}$ using the classical learner (OLS) yields an estimate

$$\hat{\theta}_c = \hat{\Sigma}_1^{-1} \hat{\mathbf{Z}}_1^\top \mathbf{Y}$$

Now, define $\Omega_t = \mathbf{R}_t^\top \hat{\mathbf{Z}}_t + \mathbf{Z}_t^\top \mathbf{R}_t$ and we have

$$
\begin{aligned}
\hat{\theta}_c &= (\Sigma_1 + \Omega_1)^{-1} (\mathbf{Z}_1 + \mathbf{R}_1)^\top \mathbf{Y} \\
&= (\Sigma_1^{-1} + \Delta_1)(\mathbf{Z}_1 + \mathbf{R}_1)^\top \mathbf{Y} \\
&= \hat{\theta}_c^* + (\Delta_1 \hat{\mathbf{Z}}_1^\top + \Sigma_1^{-1} \mathbf{R}_1^\top) \mathbf{Y}
\end{aligned}
$$

where $\Delta_1 = -\Sigma_1^{-1}\Omega_1(\Sigma_1 + \Omega_1)^{-1}$, $\hat{\theta}_c^*$ is the OLS estimate of $\theta$ for the true $\Phi$ and the second line follows from the Woodbury matrix identity. The norm of $\Delta_1$ is related to the condition number of $\Sigma_1$. The expectation of the first term is $\theta$, and the expectation of the remaining terms is the bias.

Now, we can do the same thing for the privileged estimator. Let's start with $T = 2$.

$$\begin{aligned}\hat{\theta}_p &= \hat{\Sigma}_1^{-1}\hat{\mathbf{Z}}_1^\top\hat{\mathbf{Z}}_2\hat{\Sigma}_2^{-1}\hat{\mathbf{Z}}_2^\top Y \\ &= (\Sigma_1^{-1}+\Delta_1)(\mathbf{Z}_1+\mathbf{R}_1)^\top(\mathbf{Z}_2+\mathbf{R}_2)(\Sigma_2^{-1}+\Delta_2)(\mathbf{Z}_2+\mathbf{R}_2)^\top\mathbf{Y} \\ &= \hat{\theta}_p^* + \hat{A}_1(\Sigma_2^{-1}\mathbf{R}_2^\top+\Delta_2\hat{\mathbf{Z}}_2^\top)\mathbf{Y} + (\Sigma_1^{-1}\hat{\mathbf{Z}}_1^\top\mathbf{R}_2+\Sigma_1^{-1}\mathbf{R}_1^\top\hat{\mathbf{Z}}_2+\Delta_1\hat{\mathbf{Z}}_1^\top\hat{\mathbf{Z}}_2)\hat{\beta}\ .\end{aligned}$$

Thus, the difference in bias between the two estimators is

$$\begin{aligned}\mathbb{E}[\hat{\theta}_c-\hat{\theta}_p] = &\underbrace{\mathbb{E}[\hat{\theta}_c^*-\hat{\theta}_p^*]}_{=0} \\ &+\mathbb{E}[(\Delta_1\hat{\mathbf{Z}}_1^\top+\hat{\Sigma}_1^{-1}\mathbf{R}_1^\top)\mathbf{Y}-\hat{A}_1(\Delta_2\hat{\mathbf{Z}}_2^\top+\hat{\Sigma}_2^{-1}\mathbf{R}_2^\top)\mathbf{Y} \\ &-(\Sigma_1^{-1}\hat{\mathbf{Z}}_1^\top\mathbf{R}_2+\Sigma_1^{-1}\mathbf{R}_1^\top\hat{\mathbf{Z}}_2+\Delta_1\hat{\mathbf{Z}}_1^\top\hat{\mathbf{Z}}_2)\hat{\beta}]\end{aligned}$$

More generally, we can express this difference recursively as below.

**Proposition 3.** *Let $\hat{\theta}_p \coloneqq \hat{A}_1\cdots\hat{A}_T\hat{\beta}$ be a privileged estimator using a linearly biased representation $\hat{\Phi}$, and let $\hat{A}_t^*$ be the same estimator using an unbiased representation $\Phi^*$. Then, the bias of $\hat{\theta}_p$ is*

$$\mathbb{E}[\hat{\theta}_p-\theta] = \mathbb{E}[\hat{\theta}_p-\hat{\theta}_p^*] = \mathbb{E}[E_T\hat{\beta}^* + (\hat{A}_1\cdots\hat{A}_T)(\hat{\beta}-\hat{\beta}^*)]\ ,$$

*where $E_t$ is the compounded error in transition dynamics, computed recursively as follows*

$$E_t \coloneqq (\hat{A}_1\cdots\hat{A}_t)-(\hat{A}_1^*\cdots\hat{A}_t^*) = E_{t-1}\hat{A}_t^* + (\hat{A}_1\cdots\hat{A}_{t-1})(\hat{A}_t-\hat{A}_t^*)\ ,$$

*with $E_0 = 0$. In the worst case, the bias of $\hat{\theta}_p$ grows exponentially with $T$.*

## A.4 Experiment setup & data processing

In the following we give a detailed description of the experimental setup used to obtain the results presented in Chapter 4 as well as the additional results that are shown in Appendix A.6. For a given data set, we select a combination of training set sizes and sequence length. For each unique combination of these parameters the models of interest are trained repeatedly with different random sampling. For each repetition the data is split into a train and a test set randomly before hyperparameter tuning and model training are performed. At last each model's predictions on the test set are scored by computing the coefficient of determination $R^2$. On synthetic data the test set contains 1000 samples. In the case of real-world data, where samples are limited, we test on 20% of all available data.

The preprocessing used for real-world data and the generation procedure of synthetic data is unique to each data set. We refer to the data set specific subsections for detailed descriptions of how each data set is processed. During the experiments each model uses standardized data for training and inference. To perform the data rescaling we use the StandardScaler implementation that is part of scikit-learn [55].

**Hyperparameter tuning.** The tuning of hyperparameters is carried out for each repetition and is implemented using random search and five-fold cross-validation. Each hyperparameter is sampled from a fixed interval of possible values. An overview of the ranges of different hyperparameters determined through random-search is provided in Table A.1. For the experiments with variants of generalized LuPTS we sample ten sets of hyperparameters before retraining on all training data using the best set of parameters. For the representation learners we merely sample five values for $\lambda$.

| Parameter | Description | Used in Algorithm | Value Range |
|---|---|---|---|
| $n_{RF}$ | number of random features | all random feature methods | $[0.05m, 0.8m]$ |
| $\gamma_{RRF}$ | bandwith parameter | Random ReLU methods | $[0.01, 10]$ |
| $\gamma_{RFF}$ | bandwith parameter | Random Fourier methods | $[0.001, 0.1]$ |
| $\lambda$ | loss function parameter | GRL & CRL | $[0, 1]$ |

**Table A.1:** Overview of all hyperparameter determined by hyperparameter tuning. $m$ denotes the number of samples, meaning that $n_{RF}$ is chosen from different ranges depending on the size of the training set.

**Neural network training.** The training of neural networks involves many choices and hyperparameters. We choose PyTorch's implementation [56] of the Adam optimizer [57] to train the representation learning models. If not specified otherwise the results shown in this project are obtained using a learning rate of 0.0001, a batch size of 30, leaky ReLU activations and a maximum of 1500 training epochs. In the case of neural network models, the sample sizes reported as part of the experiments denote the combined size of the training and validation set, where the validation set contains 20% of those samples. We use early stopping during the training process by keeping track of the validation loss. If a model does not improve the validation loss over a waiting period of 200 epochs we stop training early and set the network parameters to the values that obtained the lowest validation loss up until that point. In order to make sure that are results are not dependent on the specific choice of the parameters just described, we performed additional experiments. We tested different sets of training parameters to make sure our results are robust to changes in these fixed parameters.

**Generalized distillation.** In order to compare our algorithms to the alternative of using generalized distillation for privileged time series as presented by [11], we implemented a model that i) produces hypotheses of the same class as our other algorithms and ii) that adopts the learning paradigm of a student model incorporating soft targets produced by a teacher model into its loss function. For tabular data our teacher model is a multi-layer-perceptron (MLP) with $T * k$ input neurons such that all $X_t$ are used as input. It makes use of five hidden layers, each consisting of 100 neurons. In the case of the image data generated by **Clocks-LGS**, the teacher

uses an implementation of LeNet-5 on all variables $X_t$ (while sharing the encoder parameters) before concatenating the 25-dimensional output of this encoder. This combined representation is then processed by an MLP with a single hidden layer with 25 neurons. The loss function used for the student model producing the estimate $\hat{h}_{\mathrm{C}}$ is architecturally identical to the classic representation learner used in all experiments. When training the student model the mean squared error on the data and the error corresponding to the soft targets of the teacher model are combined via a hyperparameter $\lambda$:

$$\mathcal{L}_{GD} \coloneqq \lambda \frac{1}{m} \sum_{i=1}^{m} \|\hat{h}_{\mathrm{C}}(x_i) - y_i\|_2^2 + (1-\lambda) \frac{1}{m} \sum_{i=1}^{m} \|\hat{h}_{\mathrm{C}}(x_i) - \hat{h}_{\text{teacher}}(x_i)\|_2^2$$

The hyperparameter is determined via hyperparameter tuning in all repetitions as described for other hyperparameters in this section. All other training procedures follow the same logic as described above.

**Resources.** For the training of the representation learning algorithms we use a cluster of graphics processing units (GPUs) in order to reach the number of experiment repetitions required for our work. A single experiment like shown in Figure 4.4c takes several hours on 100 NVIDIA Tesla T4 GPUs. While the random feature methods do not require GPU training, they still require hyperparameter tuning which is why we compute results such as presented in Figure 4.3 on many CPU cores in parallel. While the experiments on neural networks cannot reasonably be reproduced on a single desktop machine, this is still possible within a few days for the random feature methods.

## A.5 Data set preprocessing

### A.5.1 Alzheimer progression

To test our algorithms on the task of predicting the progression of Alzheimer's disease (AD) we use an anonymized data set obtained through the Alzheimer's Disease Neuroimaging Initiative (**ADNI**) [39] under the LONI Research License. The initiative is large multi-site research study on the brains of over 2000 AD patients which collects many features such as genetic, imaging and biospecimen biomarkers. The data consists of measurements taken every 3 months with some observations missing. The outcome of interest in our experiments is the Mini Mental State Experiment (MMSE) score 48 months after the first measurement[38]. Privileged information are the measurements taken between those time points, at 12, 24 and 36 months into the program.

**Data processing.** The processing procedure used in this project is borrowed directly from the work of [10]. There is a large amount of missing information in the ADNI data set. The missingness varies with the time of when measurements were taken. Further some subjects were not present at some of the follow-up examinations. To deal with the missingness patients without an observation for the

final follow up (the outcome $Y$) are excluded from our experiments. Further, we also require that patients are present at all intermediate time steps (12, 24 and 36 months after the first measurement) which we use as privileged information. We one-hot encode categorical features and exclude features for which more than 70% of the observations are missing. To deal with the remaining missing values, mean imputation is used. Due to the filtering that we apply as a result of the missing data we only obtain 502 suitable sequences that we can use for our experiments.

### A.5.2 Traffic data

The **Traffic** data set [40] obtained through the UCI machine learning repository [58] contains hourly measurements of the traffic volume as well as weather features and holiday information. The raw data contains 48.204 records. An overview of all available features is given in Table A.2.

| Feature | Type | Description |
|---|---|---|
| Date Time | Timestamp | date and time (CST) |
| Holiday | String | name of holiday if applicable |
| Weather Description | String | brief free text description of the weather |
| Weather Main | Categorial | contains categories like clear, clouds, or rain |
| Rain_1h | Numerical | rain in $\frac{L}{h\ m^2}$ |
| Snow_1h | Numerical | snow in $\frac{L}{h\ m^2}$ |
| Temp | Numerical | temperature in Kelvin |
| Traffic Volume | Numerical | hourly reported westbound traffic volume |

**Table A.2:** Features available in the **Traffic** data set.

**Data processing.** We noticed extreme outliers in the data set as well as implausible numerical values for the temperature and rain features. Further, records for some of the hours of the timeframe (2012 - 2018) covered by this data set are missing. To deal with the extreme outliers we calculate the mean and standard deviation of each feature and remove records which contain values that are further than six standard deviations from the mean of a particular feature. We also remove a feature entirely if there is no variation left after this filtering. This is the case for the snowfall feature as snow is very rare in Minneapolis. From the date and time of each record we calculate the weekday which we add as a one-hot encoded feature and also represent the hour of the day $h \in \{0, 1, \dots 23\}$ as two separate periodic features given by

$$t_{\text{periodic}} = \left[ \sin\left(\frac{2\pi \cdot h}{24}\right),\ \cos\left(\frac{2\pi \cdot h}{24}\right) \right] . \tag{A.3}$$

This ensures that a timestamp just before midnight produces similar features compared to just after midnight. We one-hot encode the holiday information, making no difference between different types of holidays, and make this feature persist over a full calendar day. In the original data set the holiday information is only specified for the first hour of the day. The column Weather_Main contains some weather

conditions that are very rare, such as smoke and squall. As a consequence we group the different conditions before encoding them as binary variables. In particular we make drizzle, rain and squall one single feature while also grouping together fog, haze, mist and smoke as they all affect visibility.

**time series selection.** After this preprocessing that leaves only numerical values and one-hot encoded categorical values we group the data together as time series used for the experiments. In order to do so we specify a desired sequence length $T + 1$ and a sequence step size in hours. With this information we iterate through the data set assembling time series with i) no values missing ii) the correct length and step size and iii) at least a seven hour gap between each pair of sequences. The third condition is introduced to make sure one does not end up with very similar cases (for short sequences in particular) in training and test set.

### A.5.3 Square-Sign

The **Square-Sign** data set serves as a test environment for learning from privileged time series information where one can assure the conditions described by Definition 3 to hold. In particular this means creating a linear-Gaussian system which remains unobserved and combining it with an observation generating function $\psi : \mathcal{Z} \to \mathcal{X}$.

**Latent linear-Gaussian system.** The first component that makes up the generation process for **Square-Sign** (and **Clocks-LGS**) is the linear-Gaussian system which is latent, just as depicted on the right side of Figure 3.1 with $Z_t \in \mathbb{R}^d$. The first step in the data creation process is sampling each of the $d$ components of $Z_0$ from $\mathcal{N}(0, 5)$. Then the subsequent latent variables $Z_{t+1}$ are computed as

$$Z_{t+1} \coloneqq A_t^\top Z_t + \epsilon,\ \epsilon \in \mathbb{R}^d,\ \epsilon_j\ \mathcal{N}(0, 1)\ .$$

For the outcome we use the same form but with different dimensionality:

$$Y \coloneqq \beta^\top Z_T + \epsilon_y,\ \epsilon_y \in \mathbb{R}^q$$

Off-diagonal elements of the transition matrices $A_t \in \mathbb{R}^{d\times d}$ are sampled from a normal distribution $\mathcal{N}(0, 0.2)$ while the diagonal elements are set to one. In a second step we compute the spectral radius of the randomly created matrices $A_t$ via eigenvalue decomposition, obtaining the components $U\Lambda U^\top$. We then set the spectral radius to a predefined value $\lambda_{max} = 1.3$ and reassemble the matrix as

$$A_t \leftarrow U\frac{\lambda_{max}}{\lambda_s}\Lambda U^\top\ .$$

The coefficients of $\beta$ are drawn from the same normal distribution as the ones of $A_t$ but undergo no further changes.

**Observation generating function.** As the dimensionality $d$ of the latent space $\mathcal{Z}$ is not fixed we use an observation generating function that is not restricted to a specific value of $d$. For each element in $Z_t \in \mathcal{Z} = \mathbb{R}^d$ we create two elements

in $X_t \in \mathcal{X} = \mathbb{R}^{2d}$ by denoting its sign separately from its square. This gives the following nonlinear observation generating function:

$$X_t \coloneqq \psi(Z_{(t)}) = [Z_{(t,1)}^2, \operatorname{sgn}(Z_{(t,1)}), \ldots, Z_{(t,d)}^2, \operatorname{sgn}(Z_{(t,d)})]^\top$$

### A.5.4 Clocks-LGS

This data set serves the purpose of testing our algorithms on a different modality with high dimensional data. In particular the idea was to use image data as this is a domain where neural networks have been very successful. For this reason we combine a latent dynamical system with an image generation process which we explain in detail in this section.

**Latent linear-Gaussian system.** We use exactly the same setup as we do for the **Square-Sign** latent dynamical system as described in Section A.5.3. The only difference here is the dimensionality of the latent variables, transition matrices and the outcome. For **Clocks-LGS** we generally have $d = 2$ and $q \in \{1, 2\}$.

**Image generation.** The second part of **Clocks-LGS** is creating images from two dimensional latent vectors $Z_t = [Z_t^{(1)}, Z_t^{(2)}]^\top$. The goal was to keep it the process simple while using small black and white images of 28×28 pixels. In addition we wanted each image to have no ambiguity with respect to the latent state it represents. We represent the first component by a clock hand mounted at the image center. One can think of $Z_t^{(1)}$ as the angle in radian, meaning the hand points straight up for $Z_t^{(1)} = 0$ or $Z_t^{(1)} = 2\pi$ and straight down for $Z_t^{(1)} = \pi$. To visualize a full rotation we increase the size of the cirle around the image center in discrete steps for each mutliple of $2\pi$. For negative values the circle is empty (black) while it is filled (white) for positive values. For the second component we make use of the same logic but instead of a clock hand, we only use a circle that orbits the image center. The two hands cannot obscure each other as the orbiting cirle uses a larger radius.

### A.5.5 Air quality

Due to health concerns the air quality in Chinese cities has become an important topic. The $\mathbf{PM}_{2.5}$ data set contains hourly meteorologic information and the concentration of small particles ($\mathrm{PM}_{2.5}$) for the cities Beijing, Shanghai, Guangzhou, Chengdu and Shenyang [59]. The individual features available for all cities are listed in Table A.3. In addition to the features listed, the data includes the date and time of each record. Just like in the preprocessing of **Traffic** we compute a periodic time feature using expression A.3 to represent the time of day of each record. For each numerical feature we calculate the mean and standard deviation and remove rows with values that are more extreme than six standard deviations from the mean. We also remove rows with missing categorical features, which are then represented as one-hot vectors. Apart from differences in the preprocessing we consider the same prediction task as [10] which is predicting the future particle concentration for a fixed time horizon given current observations.

| Feature | Type | Description |
|---|---|---|
| season | Numerical | season (1 to 4) of the data in this row |
| PM | Numerical | $PM_{2.5}$ particle concentration in $\mu g/m^3$ |
| DEWP | Numerical | dew point in °C |
| TEMP | Numerical | air temperature in °C |
| HUMI | Numerical | humidity in % |
| PRES | Numerical | atmospheric pressure in hPa |
| cbwd | Categorical | combined wind direction in {N,W,S,E, NW, SW, NE, SE} |
| Iws | Numerical | cumulated wind speed in $m/s$ |
| precipitation | Numerical | hourly precipitation in $mm$ |
| Iprec | Numerical | cumulated precipitation in $mm$ |

**Table A.3:** Features available as part of the $\mathbf{PM}_{2.5}$ data set on the air quality of five large Chinese cities.

## A.6 Additional experiment results

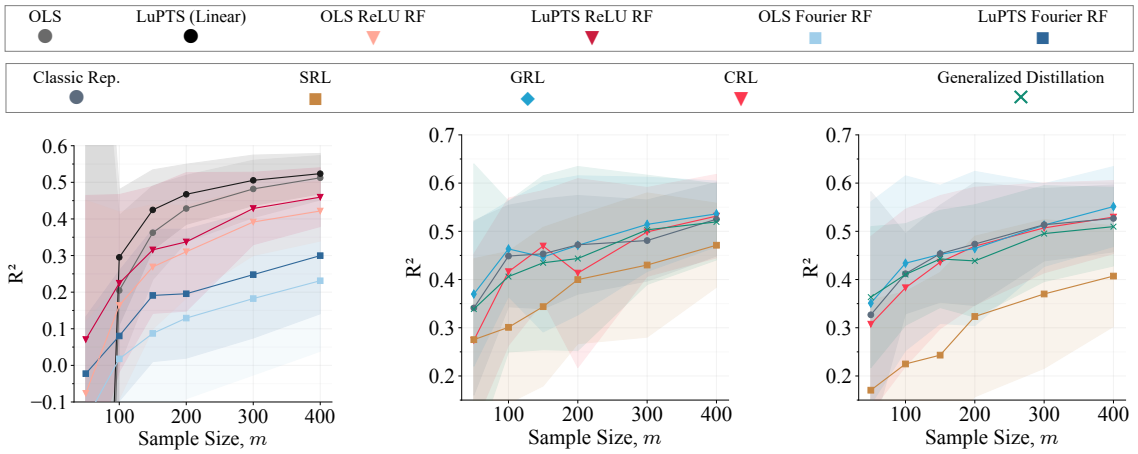
In the course of this section we present a larger scope of our experimental results. We demonstrate the predictive accuracy of the algorithms introduced in Chapter 3 in terms of the mean coefficient of determination $R^2$ over different settings on five data sets. The variation of the results over repetitions is represented by the shaded areas in the visualizations, which denotes one standard deviation above and below the mean value.

### A.6.1 Two regimes of generalized LuPTS

As seen in Figure 3.3 and demonstrated by Proposition 1, generalized LuPTS becomes equivalent to the corresponding classical learner when the number of features $\hat{d}$ is larger than the number of samples $m$. In the following we provide the experiment details that led to Figure 3.3.

In order to evaluate the dependency on the number of features we used linear LuPTS and OLS on a synthetically generated linear-Gaussian system as displayed on the left of Figure 3.1. We use the same setup as for the latent dynamics in **Square-Sign** but without a nonlinear observation generating function. For each number of features $k = \hat{d} = d$ we sample 50 such systems with different dynamics, each producing a training and test set with fixed sample size $m = 100$ and test set size of 1000 samples. The systems are all configured with $T = 3$ and $q = 10$. We train and score both estimators on all of the data generating systems before computing the mean coefficient of determination over all systems with the same number of features.

### A.6.2 Alzheimer progression



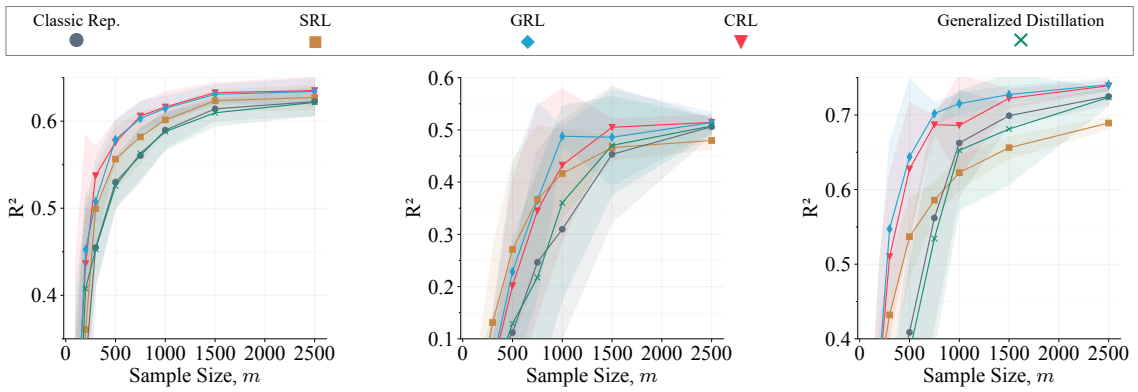

**(a)** Generalized LuPTS on the reduced setting, $T = 2$.

**(b)** Representation learners and generalized distillation on the reduced setting, $T = 2$.

**(c)** Representation learners and generalized distillation on the full setting, $T = 4$.

**Figure A.1:** Generalized LuPTS and the representation learning algorithms tested in terms of their predictive accuracy on different settings of the **ADNI** prediction task. Each experiments are based on 25 repetitions while the random feature experiment consists of 60 repetitions.

### A.6.3 Clocks_LGS



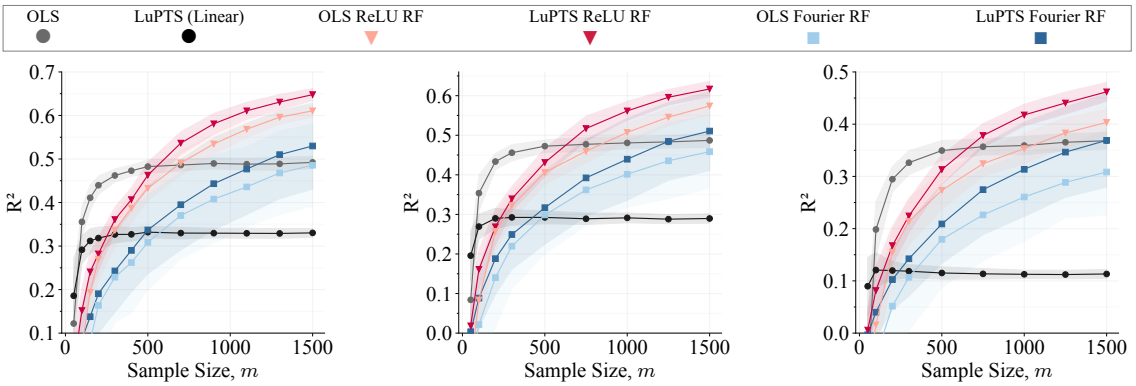

**(a)** Sequences of length three, one privileged time point, $T = 2$.

**(b)** Sequences of length four, two privileged time points, $T = 3$.

**(c)** Sequences of length six, four privileged time points, $T = 5$.

**Figure A.2:** Prediction accuracy of the representation learning algorithms and generalized distillation on **Clocks-LGS** with a two dimensional output and varying sequence lengths. Each experiment is based on 25 repetitions.

### A.6.4 Square-Sign

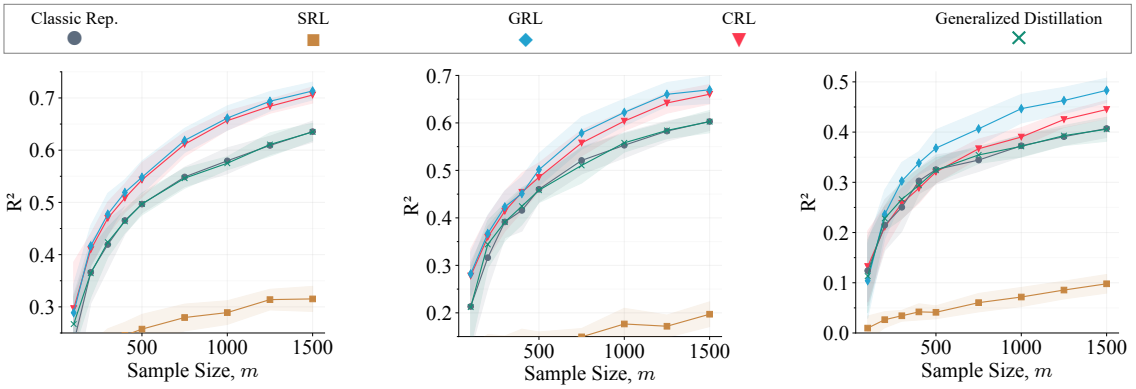



**(a)** Sequences of length six, four privileged time points, $T = 5$.

**(b)** Sequences of length seven, five privileged time points, $T = 6$.

**(c)** Sequences of length eight, six privileged time points, $T = 7$.

**Figure A.3:** Predictive accuracy ($R^2$) of the different variants of generalized LuPTS applied to the prediction task offered by **Square-Sign**. The DGP was configured with different sequence lengths and the experiments represent 60 repetitions.



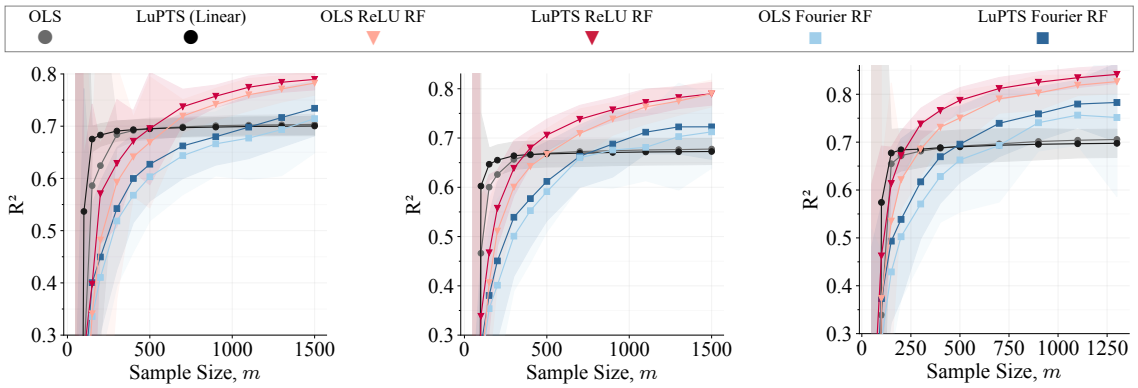

**(a)** Sequences of length six, four privileged time points, $T = 5$.

**(b)** Sequences of length seven, five privileged time points, $T = 6$.

**(c)** Sequences of length eight, six privileged time points, $T = 7$.

**Figure A.4:** Predictive accuracy of the different representation learners introduced in Section 3.4 when applied to the **Square-Sign** data set for different sequence lengths. The experiments are based on 25 repetitions.

### A.6.5 Traffic data

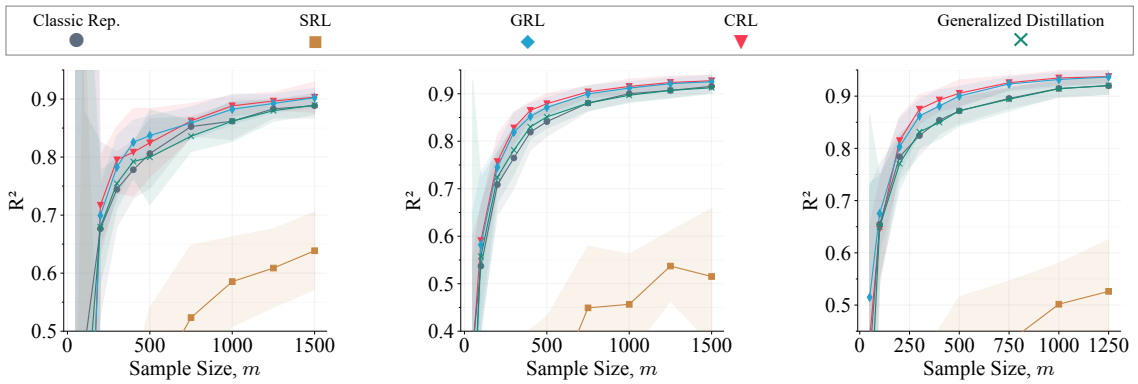



**(a)** Sequences of length three, one privileged time point, $T = 2$.

**(b)** Sequences of length four, two privileged time points, $T = 3$.

**(c)** Sequences of length five, three privileged time points, $T = 4$.

**Figure A.5:** Prediction accuracy of the different variants of generalized LuPTS on **Traffic** with varying sequence lengths. Based on 60 repetitions and four hour steps in each time series.



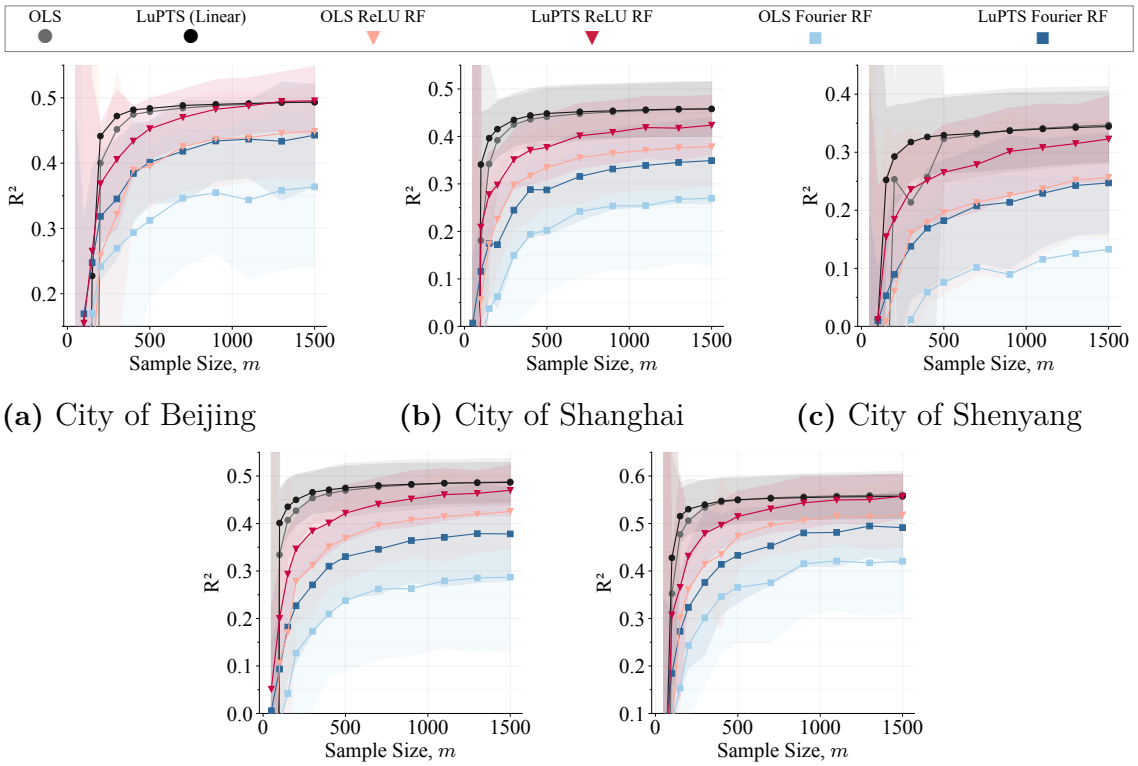

**(a)** Sequences of length three, one privileged time point, $T = 2$.

**(b)** Sequences of length four, two privileged time points, $T = 3$.

**(c)** Sequences of length five, three privileged time points, $T = 4$.

**Figure A.6:** Prediction accuracy of the different representation learning algorithms and generalized distillation on **Traffic** with varying sequence lengths. Based on 25 repetitions and four hour steps in each time series.

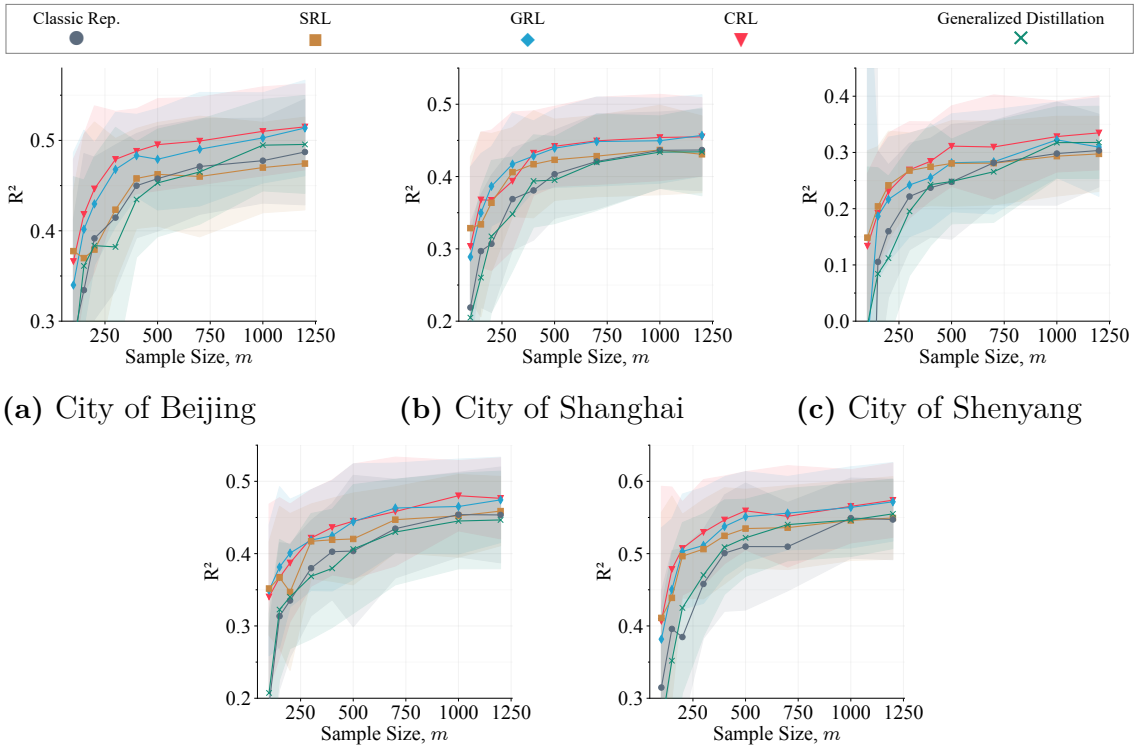
### A.6.6 Air quality




**(a)** City of Beijing **(b)** City of Shanghai **(c)** City of Shenyang

**(d)** City of Guangzhou **(e)** City of Chengdu

**Figure A.7:** Predictive accuracy of the different generalized LuPTS variants on the prediction task posed by the $\mathbf{PM}_{2.5}$ data set. All experiments use time series of length five ($T = 4$), where each time step is two hours long. The results were computed based on 60 repetitions.




**(a)** City of Beijing **(b)** City of Shanghai **(c)** City of Shenyang

**(d)** City of Guangzhou **(e)** City of Chengdu

**Figure A.8:** Evaluation of the sample efficiency of the represenation learning algorithms of Section 3.4 on the $\mathbf{PM}_{2.5}$ air quality prediction task. All experiments use time series of length five ($T = 4$), where each time step represents two hours. The results were computed based on 25 repetitions.