Merging Two Cultures: Deep and Statistical Learning

Anindya Bhadra[*], Jyotishka Datta[†], Nick Polson[‡], Vadim Sokolov[§] and Jianeng Xu[¶]

Article Type:

Advanced Review

Abstract

Our goal is to provide a review of deep learning methods which provide insight into structured high-dimensional data. Merging the two cultures of algorithmic and statistical learning sheds light on model construction and improved prediction and inference, leveraging the duality and trade-off between the two. Prediction, interpolation, and uncertainty quantification can be achieved using probabilistic methods at the output layer of the model. Rather than using shallow additive architectures common to most statistical models, deep learning uses layers of semi-affine input transformations to provide a predictive rule. Applying these layers of transformations leads to a set of attributes (or, features) to which probabilistic statistical methods can be applied. Thus, the best of both worlds can be achieved: scalable prediction rules fortified with uncertainty quantification where sparse regularization finds the features. We review the duality between shallow and wide models such as principal components regression, and partial least squares and deep but skinny architectures such as autoencoders, multilayer perceptrons, convolutional neural net, and recurrent neural net. The connection with data transformations is of practical importance for finding good network architectures. By incorporating probabilistic components at the output level, the predictive uncertainty is allowed. We illustrate this idea by comparing plain Gaussian Processes (GP) with Partial Least Squares + Gaussian Process (PLS+GP) and Deep Learning + Gaussian Process (DL+GP).

[*]Purdue University, bhadra@purdue.edu
[†]Virginia Tech, jyotishka@vt.edu
[‡]U Chicago. ngp@chicagobooth.edu
[§]George Mason. vsokolov@gmu.edu
[¶]U Chicago. jianeng.xu@chicagobooth.edu

1 GRAPHICAL TABLE OF CONTENTS

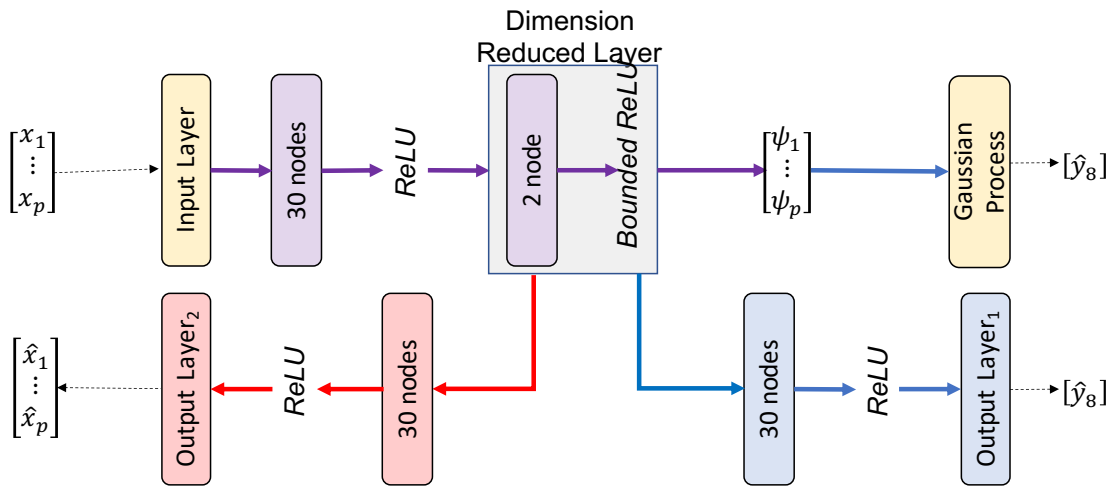


Figure 1: Architecture of the DL-GP model, an example of the synergy between algorithmic and statistical learning, leveraging probabilistic techniques at the output layer to achieve scalable prediction rules with uncertainty quantification.

2 INTRODUCTION

Deep learning is an emerging methodology for analysis of large, high-dimensional data sets. Our goal is to provide a brief overview of deep learning methods through a lens of statistical modeling. In his seminal paper, Breiman [2001] highlighted the contrast between an algorithmic approach and traditional statistical modeling ('data modeling') for 21^{st} century data analytics. In his view, data modeling approaches start with a simpler stochastic model for the data generating process, while algorithmic modeling approaches assume the generating process is 'complex and unknown' and focus their effort on understanding high-dimensional data structure.

The statistical learning approach involves identifying a function that maps a high-dimensional input variable $\mathbf{x} \in \mathbb{R}^p$ to an output variable $\mathbf{y} \in \mathbb{R}^q$,

$$\mathbf{y} = F(\mathbf{x}) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \text{ is the random error.}$$

The map $F(\cdot)$ is to be learned from the observed matrices of inputs, denoted by $\mathbf{X} \in \mathbb{R}^{n \times p}$

and corresponding outputs $\mathbf{Y} \in \mathbb{R}^{n \times q}$. The main question is how to learn the nonlinear function $F(\cdot)$. Recently, tree-based and deep learning methods have been shown to have good empirical performance on a variety of tasks. The statistical modeling approach makes uncertainty quantification paramount and, following Breiman [2001], we write:

$$\text{output} = F(\text{predictor variables}, \text{parameters}, \text{error}).$$

The main limitations of statistical modeling stem from the difficulties in model specification and validation, particularly in higher dimensions.

Deep learners, on the other hand, are based on the superposition of univariate affine functions [N. G. Polson & Sokolov, 2017] and are universal approximators Sutskever & Hinton [2008]. Whilst Gaussian Processes (GP) [Gramacy & Lee, 2008, Higdon et al., 2008] are also universal approximators and can capture relations of high complexity, they suffer in high-dimensional settings due to the behavior of distance metrics and the exponential growth of the number of design points needed [Binois & Wycoff, 2021]. Similarly, hierarchical models are flexible stochastic models but require high-dimensional integration and computationally expensive sampling procedure. On the other hand, tree-based methods can be very effective in high-dimensional problems, as they do not need many tuning parameters and can proceed without a formal variable selection process [Cutler et al., 2008]. Similarly, deep learning is based on fast gradient learning algorithms such as stochastic gradient descent (SGD) and its variants made scalable with modern computational techniques like automated differentiation (AD) and accelerated linear algebra (XLA), available within TensorFlow or PyTorch. DL methods avoid the *curse of dimensionality* by pattern matching and using interpolation to predict other regions of the input space [Poggio et al., 2017]. Much of the successes of DL can be attributed to very efficient ways of calculating the gradient, building on early successes of backpropagation [Rumelhart et al., 1995], unlike Gausian processes or nonparametric models.

There are many successful applications of deep learning across many fields, including speech recognition, translation, image processing, robotics, engineering, data science, healthcare among many others. These applications include solving three main tasks:

(a) Image Processing and Classification [Esteva et al., 2021, Ronneberger et al., 2015, L. Li et al., 2020, Shallue & Vanderburg, 2018]

(b) Natural Language Processing [Oord et al., 2016, Brown et al., 2020, Devlin et al., 2018, Vaswani et al., 2017]

(c) Robotics and Reinforcement Learning [Dean, 2020, Agarwal et al., 2021, Rigter et al., 2021]

In the statistical realm, deep learning has been applied to spatio-temporal modeling [N. G. Polson & Sokolov, 2017, McDermott & Wikle, 2019, Jacquier et al., 2021], finance [Dixon et al., 2019, Heaton et al., 2017, Gu et al., 2021], time series Lim et al. [2021], Alaa & van der Schaar [2019], Makridakis et al. [2020], Salinas et al. [2020], Triebe et al. [2021], tabular data Borisov et al. [2021], Popov et al. [2019] among others. Given the success of deep learning thus far, the scope of applications is likely to grow over the coming years. In this paper, we argue that deep learning (abbreviated as DL, henceforth) has wide applicability to traditional statistical areas for tabular data structures including categorical, spatial and time series analysis. Until now, traditional statistical models have relied heavily on additive functions with low approximation capacity based on shallow architectures. From a statistical viewpoint, much attention has been paid to stochastic models that combine with the deterministic part of a statistical model.

The algorithmic modeling culture has achieved much success in high-dimensional problems, pushing the boundary of accuracy in object detection, speech recognition, image processing and countless other important applications. The spearhead of algorithmic modeling culture, DL assumes a very flexible class of predictors, $F(\mathbf{X})$, and directly trains the model using a predictive mean squared error loss. Such classes of functions include decision trees and neural networks with multiple layers. The goal is to find a predictor rule (*aka* algorithm), $F(\mathbf{X})$, to evaluate on $\mathbf{X}$ to predict output $\mathbf{Y}$. The caveat with an algorithmic approach is that it lacks straightforward *uncertainty quantification*.

Uncertainty Quantification (UQ). Statistical models are capable of representing uncertainty in predictions and parameters, but when input-output relations are modeled using deterministic functions like neural network, this property is lost. Understanding and characterizing uncertainty is key in many science and engineering applications. Neural network can be viewed as an essentially nonparametric regression procedure which uses sparse or skinny representation [Olshausen & Field, 1996]. Bishop [1995] considers neural networks from a

statistical modeling point of view and presents neural network as an extension to more traditional functions used in statistical modeling. Ripley [2007] presents a feed-forward neural network as a way to project input into lower dimensional space within which the approximation can be preferred, and compares it to projection pursuit regression [Huber, 1985]. We can, thus, think of a neural network as a way to generate a representation of data in a different subspace, where each layer is a representation [Wang & Rocková, 2020, Fan et al., 2021].

Our main contribution in this paper is to provide a much-needed insight into the duality and trade-off between shallow-and-wide statistical methods (*viz.* Principal Components Analysis, Principal Components Regression, Partial Least Squares) and the deep-and-skinny DL architecture (*viz.* feed-forward ReLU, convolutional layers, auto-encoders, and recurrent layers) in the light of two critical tasks: dimensionality expansion and dimensionality reduction, and argue how *merging* might benefit both worlds. To illustrate these ideas, we demonstrate two merged methods: DL-PLS and PLS-GP in Section 5 via real and synthetic examples. The rest of the paper is outlined as follows. Section 3 reviews the two cultures of statistical modeling and deep learning and describes classes of deep learners vis-á-vis unsupervised and supervised dimension reduction methods, such as principal components (Section 3.4) and partial least squares (Section 3.4). Section 4 discusses deep learning architectures for dimensionality reduction of high-dimensional data and introduces DL-PLS and PLS-GP. Section 6 provides applications in regression, classification and prediction, and Section 7 concludes by providing some directions of future investigations.

3 Dimensionality Reduction and Expansion

3.1 General latent feature model

Given a training data-set of input-output pairs $(\mathbf{X}_i, \mathbf{Y}_i)_{i=1}^N$, the goal is to find a prediction rule for a new output $\mathbf{Y}_*$ given a new input $\mathbf{X}_*$. Let $\mathbf{Z}$ denote latent hidden features that are to be hand-coded or learned from the data and our nonlinear latent feature predictive

model takes the form:

$$\mathbf{Y} \mid \mathbf{Z} \sim p(\mathbf{Y} \mid \mathbf{Z}) \tag{1}$$

$$\mathbf{Z} = \phi(\mathbf{X}) \tag{2}$$

where $\phi(\cdot)$ is a data transformation that allows for relations between latent features $\mathbf{Z} = \phi(\mathbf{X})$ and $\mathbf{Y}$ to be modeled by a well-understood probabilistic model p. Typically, $\phi(\cdot)$ will perform dimension reduction or dimension expansion and can be learned from data. It is worthwhile to emphasize that the top level of such a model is necessarily stochastic.

As pointed out before, the basic problem of machine learning is to learn a predictive rule from observed pairs $(\mathbf{X}, \mathbf{Y})$, $\mathbf{Y}_\star = F(\mathbf{X}_\star)$ where $F(\mathbf{X}_\star) = \mathbb{E}\{\mathbf{Y} \mid \phi(\mathbf{X}_\star)\}$. Even though it is well known that deep learners are universal approximators, it is still an open area of research to understand why deep learners generalize well on out-of-sample predictions. One of the important factors for the success of deep learning approach is the ability to perform a non-linear dimensionality reduction [Goodfellow et al., 2016].

The purely statistical approach requires full specification of the conditional distribution $p(\mathbf{Y} \mid \mathbf{X})$ and then uses conditional probability via Bayes' rule to perform inference. However, a fully Bayesian approach is often computationally prohibitive in high-dimensional feature space, without resorting to approximations or foregoing UQ. The alternative approach is to perform a data transformation or reduction on the data input $\mathbf{X}$, such as performing a PCA or PCR first and then use a statistical approach on the transformed feature space. Deep learning methods can fit into this spectrum by viewing it as a non-linear PCA or PLS [N. Polson et al., 2021, Tran et al., 2019, Malthouse et al., 1997]. However, it possesses some unique properties not available for shallow models.

3.2 Deep Learning Expansions

Similar to a tree model that finds features (*aka* tree leaves) via recursive space partitioning, the deep learning model finds the regions by using hyperplanes at the first layer and combinations of hyperplanes in the further layers. The prediction rule is embedded into a parameterized deep learner, a composite of univariate semi-affine functions, denoted by $F_{\mathbf{W}}$ where $\mathbf{W} = [\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}]$ represents the weights of each layer of the network. A deep

learner takes the form of a composition of link functions:

$$F_{\mathbf{W}} = f_L \circ f_{L-1} \circ \cdots \circ f_1 \text{ where } f_L = \sigma_L(\mathbf{W}_L \phi(\mathbf{X}) + b_L),$$

where $\sigma_L(\cdot)$ is a univariate link or activation function. Specifically, let $\mathbf{Z}^{(l)}$ denote the l^{th} layer, and so $\mathbf{X} = \mathbf{Z}^{(0)}$. The final output is the response $\mathbf{Y}$, which can be numeric or categorical. A deep prediction rule is then $\hat{\mathbf{Y}}(\mathbf{X}) = \mathbf{W}^{(L)}\mathbf{Z}^{(L)} + \mathbf{b}^{(L)}$ where

$$\begin{aligned}
\mathbf{Z}^{(L)} &= f^{(L)}\left(\mathbf{W}^{(L-1)}\mathbf{Z}^{(L-1)} + \mathbf{b}^{(L-1)}\right), \\
&\cdots \\
\mathbf{Z}^{(2)} &= f^{(2)}\left(\mathbf{W}^{(1)}\mathbf{Z}^{(1)} + b^{(1)}\right), \\
\mathbf{Z}^{(1)} &= f^{(1)}\left(\mathbf{W}^{(0)}\phi(\mathbf{X}) + \mathbf{b}^{(0)}\right).
\end{aligned}$$

It is often beneficial to replace the original input $\mathbf{X}$ with the features $\mathbf{Z} = \phi(\mathbf{X})$ of lower dimensionality when developing a predictive model for $\mathbf{Y}$. For example, in the context of regressions, a lower variance prediction rule can be obtained in lower dimensional space (see Lindley [1968], for a fully Bayesian discussion). DL simply uses a composition or superposition of semi-affine filters (*aka* link functions) Tran et al. [2019], leading to a new framework for high-dimensional modeling in Section 3.

Deep learning can then be viewed as a feature engineering solution and one of finding nonlinear factors via supervised dimension reduction. A composition of hand-coded characteristics i.e. dimension expanding, with supervised learning of data filters i.e. dimension reduction. Advances in computation allow for massive data and gradients of high-dimensional nonlinear filters. Neural networks can be viewed from two perspectives: either as a flexible link function, as in a generalized linear model [McCullagh, 2019], or as a method to achieve dimensionality reduction, similar to sliced inverse regression [K.-C. Li, 1991] or sufficient dimensionality reduction [Cook & Forzani, 2009].

One advantage of *depth* is that the hierarchical mixture allows the width of a given layer to be manageable [Goodfellow et al., 2016]. With a single layer (*e.g.*, kernel PCA/SVM) we need exponentially many more basis functions in that layer. Consider kernel PCA with say radial basis functions (RBF) kernels: technically there are infinitely many basis functions, but it cannot handle that many input dimensions. Presumably, a deep neural network allows

a richer class of covariances that allows anisotropy and non-stationarity. In the end, this is reflected in the function realizations from a DNN. To see this, consider the deep GP models, which are infinite width limits of DNNs [Damianou & Lawrence, 2013, Fei et al., 2018]. There is a recursive formula connecting the covariance of layer k to that of layer $k+1$, but no closed form. The covariance function of the final hidden layer is probably very complicated and capable of expressing an arbitrary number of features, even if the covariances in each layer may be simple. The increase in dimensionality happens through the hierarchical mixture, rather than trying to do it all in one layer. From a statistical viewpoint, this is similar to the linear shallow wide projections introduced by [Wold, 1975] and the sufficient dimension reduction framework of Cook [2007].

In the context of unsupervised learning, information in the marginal distribution, $p(\mathbf{X})$, of the input space is used as opposed to the conditional distribution, $p(\mathbf{X} \mid \mathbf{Y})$. Methods such as Principal Components Analysis(PCA), Principal Components Regression(PCR), Reduced Rank Regression (RRR), Projection-Pursuit Regression (PPR) all fall into this category and Partial Least Squares(PLS), Sliced Inverse Regression (SIR) are examples of supervised learning of features, see N. G. Polson & Sokolov [2017] for further discussion.

We first uncover the structure in the predictors relevant for modeling the output $\mathbf{Y}$. The learned factors are denoted by $F(\phi(\mathbf{X}))$ and are constructed as a sequence of input filters. The predictive model is given by a probabilistic model of the form $p(\mathbf{Y} \mid \mathbf{X}) \equiv p(\mathbf{Y} \mid F(\phi(\mathbf{X})))$. Here $\phi : \mathbb{R}^p \mapsto \mathbb{R}^c,\ c \gg p$ initially expands the dimension of the input space by including terms such as interactions, dummy variables (*aka* one-hot encodings) and other nonlinear features of the input space deemed relevant. Then, F reduces dimension of deep learning by projecting back with a univariate activation function into an affine space (*aka* regression). This framework also sheds light on how to build deep (skinny) architectures. Given n data points, we split into $L = 2^p$ regions [Harding, 1967] so that there is a *fixed* sample size within each bin. This process transforms $\mathbf{X}$ into many interpretable characteristics. This can lead to a huge number of predictors that can be easily dealt within the DL architecture, making the advantage of *depth* clear.

3.3 Dimensionality Expansion

First, we review 'dimensionality expansions': data transformation that transforms an input vector $\mathbf{X}$ into a higher dimensional vector $\phi(\mathbf{X})$. One approach is to use hand-coded predictors. This expanded set can include terms such as interactions, dummy variables or nonlinear functional of the original predictors. The goal is to model the joint distribution of outputs and inputs, namely $p(\mathbf{Y}, \phi(\mathbf{x}))$, where we allow our stochastic predictors.

Kernel Expansion: The kernel expansion idea is to enlarge the feature space via basis expansion. The basis is expanded using nonlinear transformations of the original inputs:

$$\phi(\mathbf{X}) = (\phi_1(\mathbf{X}), \phi_2(\mathbf{X}), \ldots, \phi_M(\mathbf{X}))$$

so that linear regression $\hat{\mathbf{Y}} = \phi(\mathbf{X})^T\boldsymbol{\beta} + \boldsymbol{\beta}_0$ or generalized linear model can be used to model the input-output relations. Here, the 'kernel trick' increases dimensionality, and allows hyperplane separation while avoiding an exponential increase in the computational complexity. The transformation $\phi(\mathbf{x})$ is specified via a kernel function $K(\cdot, \cdot)$ which calculates the dot product of feature mappings: $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$. By choosing a feature map ϕ, we implicitly choose a kernel function and, conversely, every positive semi-definite kernel matrix corresponds to a feature mapping ϕ. For example, when $\mathbf{X} \in \mathbb{R}^2$, choosing $K(\mathbf{X}, \mathbf{X}') = (1 + \mathbf{X}^T\mathbf{X}')^2$ is equivalent to expanding the basis to $\phi(\mathbf{X}) = (1, \sqrt{2}\mathbf{X}_1, \sqrt{2}\mathbf{X}_1, \mathbf{X}_1^2, \mathbf{X}_2^2, \sqrt{2}\mathbf{X}_1\mathbf{X}_2)$ [see e.g. Hastie & Tibshirani, 2009, Ch. 12].

Tree Expansion: Similar to kernels, we can think of trees as a technique for expanding a feature space. Each region in the input space defined by a terminating node of a tree corresponds to a new feature. Then, the predictive rule becomes very simple: identify in which region the new input is and use the average across observations or a majority voting rule from this region to calculate the prediction.

3.4 Dimensionality Reduction: PCA, PCR and PLS

Given input/predictors $\mathbf{X}$ and response $\mathbf{Y}$ and associated observed data $\mathbf{X} \in \mathbb{R}^{n\times p}$ and $\mathbf{Y} \in \mathbb{R}^{n\times q}$, the goal is to find data transformations $(\mathbf{Y}, \mathbf{X}) \mapsto \phi(\mathbf{Y}, \mathbf{X})$ so that modeling the transformed data becomes an easier task. In this paper, we consider several types of transformations and model non-linear relations.

We start by reviewing the widely used singular value decomposition (SVD) which allows finding linear transformations to identify a lower dimensional representation of either $\mathbf{X}$, by what is known as principal component analysis (PCA), or, when using both $\mathbf{X}$ and $\mathbf{Y}$, known as partial least squares (PLS). First, start with the SVD decomposition of the input matrix: $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{W}^T$, which is full-rank if $n > p$. Here, $\mathbf{D} = \text{diag}(d_1, \ldots, d_p)$ are the nonzero ordered singular values ($d_1 \geq \ldots \geq d_p$) . The matrices $\mathbf{U}$ and $\mathbf{W}$ are orthogonal matrices of dimensions $n \times p$ and $p \times p$ with columns of $\mathbf{U}$ as the right singular vectors and columns of $\mathbf{W}$ as the left singular vectors, $\mathbf{W}$ can be also thought of as the matrix consisting of eigenvectors for $\mathbf{S} = \mathbf{X}^T\mathbf{X}$. We can then transform the original first layer to an orthogonal regression, namely defining $\mathbf{Z} = \mathbf{U}\mathbf{D}$ whose columns are the principal components. For PCR, using $\boldsymbol{\alpha} = \mathbf{W}^T\boldsymbol{\beta}$, we arrive at the corresponding OLS estimator $\hat{\boldsymbol{\alpha}} = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{Y} = \mathbf{D}^{-1}\mathbf{U}^T\mathbf{y}$, and obtain $\hat{y}_{pcr} = \sum_{j=1}^{K} \hat{\alpha}_j \mathbf{z}_j$, where $\hat{\alpha}_j = \mathbf{z}_j^T\mathbf{y}/\mathbf{z}_j^T\mathbf{z}_j$, since $\mathbf{z}_j$'s are orthogonal, and $K \ll p$ denotes the reduced dimension that captures a certain percentage of the total variability.

PCR, as an unsupervised approach to dimension reduction, has a long history in statistics [Massy, 1965]. Specifically, we first center and standardize $(\mathbf{Y}, \mathbf{X})$, followed by a singular value decomposition of $\mathbf{V} := \text{ave}(\mathbf{X}\mathbf{X}^T) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{X}_i\boldsymbol{X}_i^T$ where $\text{ave}(\cdot)$ denotes the empirical average. Then, we find the eigenvalues e_j^2 and eigenvectors arranged in non-increasing order, so we can write:

$$\mathbf{V} = \sum_{j=1}^{p} e_j^2 \boldsymbol{v}_j \boldsymbol{v}_k^T.$$

This leads to a sequence of regression models $(\hat{Y}_0, \ldots, \hat{Y}_K)$ with $\hat{Y}_0$ being the overall mean:

$$\hat{Y}_L = \sum_{l=0}^{K} (\text{ave}(W_l^T\boldsymbol{X})/e_l^2)\boldsymbol{v}_l^T\boldsymbol{X}.$$

Therefore, PCR finds features $\{\mathbf{Z}_k\}_{k=0}^K = \{\boldsymbol{v}_k^T\boldsymbol{x}\}_{k=0}^K = \{\boldsymbol{f}_k\}_{k=0}^K$.

PLS and SVD Algorithm: Partial least squares, or PLS, is a related dimension reduction technique similar to PCR that first identifies a lower-dimensional set of features and then fits a linear model on this feature set, but PLS does this in a supervised fashion unlike PCR. In successive steps, PLS finds a reduced dimensional representation of $\mathbf{X}$ that is relevant for the response $\mathbf{Y}$.

PCA and multivariate output: Principal Components Analysis requires us to compute a reduction of multivariate output $\mathbf{Y}$ using a singular value decomposition of $\mathbf{Y}$ by finding

eigenvectors of $\mathbf{Z} = \text{ave}(\mathbf{Y}\mathbf{Y}^T)$. Then, the output is a linear combination of the singular vectors

$$\mathbf{Y} = \mathbf{W}_1\mathbf{Z}_1 + \cdots + \mathbf{W}_k\mathbf{Z}_k,$$

where the weights $\mathbf{W}_i$ follow a Gaussian Process, $\mathbf{W} \sim \text{GP}(m, K)$. Hence, the method can be highly non-linear. This method is typically used when input variables come from a designed experiment. If the interpretability of factors is not important, and from a purely predictive point of view, PLS will lead to improved performance.

In the light of the above, one can view deep learning models as non-stochastic hierarchical data transformations. The advantage is that we can learn deterministic data transformations before applying a stochastic model. This allows us to establish a connection between a result due to Brillinger [2012] and the use of deep learning models to develop a unified framework for modeling complex high-dimensional data sets. The prediction rule can be viewed as interpolation. In high-dimensional spaces, one can mix-and-match the deterministic and stochastic data transformation rules.

Partial Least Squares (PLS): Partial Least Sqaures or PLS is one of the first statistical methods for prediction in high-dimensional linear regression [Wold, 1985, Cook & Forzani, 2018], where the number of variables p need not be large compared to the sample size n. Helland [1990] defines the PLS regression models and Frank & Friedman [1993] gives a shrinkage interpretation of the prediction rule. Naik & Tsai [2000] demonstrated that the PLS regression provides a consistent estimator when the predictors follow a single-index model with $n \to \infty$ and p fixed. Delaigle & Hall [2012] extend this work to functional data. Chun & Kelecs [2010] warn of inconsistencies in cases when, $p/n \to 0$ and they use this to motivate a sparse PLS regression. They study a single-component PLS and found $\sqrt{n}$ - consistency as $n, p \to \infty$ in a number of reasonable settings. Cook & Forzani [2018] extend this to multiple input regressions and regression.

PLS uses both $\mathbf{X}$ and $\mathbf{Y}$ to calculate the projection, and simultaneously finds projections for both input $\mathbf{X}$ and output $\mathbf{Y}$, making it suitable for problems with high-dimensional output vector as well as input vector. PLS finds a projection direction that maximize covariance between $\mathbf{X}$ and $\mathbf{Y}$, the resulting projections $\mathbf{U}$ and T for $\mathbf{Y}$ and $\mathbf{X}$, respectively are called score matrices, and the projection matrices $\mathbf{P}$ and $\mathbf{Q}$ are called loadings. The $\mathbf{X}$-score

matrix T, has L columns, one for each feature and $\mathbf{L}$ is chosen via cross-validation. The key principle is that T is a good predictor of $\mathbf{U}$, the $\mathbf{Y}$-scores. These relations are summarized as: $\mathbf{Y} = \mathbf{UQ} + \boldsymbol{E}$, $\mathbf{X} = \mathrm{T}\mathbf{P} + \mathbf{F}$. Here $\mathbf{Q}$ and $\mathbf{P}$ are orthogonal projection (loading) matrices and T and $\mathbf{U}$ are projections of $\mathbf{X}$ and, $\mathbf{Y}$ respectively. Helland [1990] showed that the PLS estimator can be calculated sequentially as:

$$\hat{\mathbf{B}}_{\mathrm{PLS}} = \mathbf{R}(\mathbf{R}^T\mathbf{S}_{XX}\mathbf{R})^{-1}\mathbf{R}^T\mathbf{s}_{XY},$$

where $\mathbf{R} = (\mathbf{s}_{XY}, \mathbf{S}_{XX}\mathbf{s}_{XY}, \ldots, \mathbf{S}_{XX}^{q-1}\mathbf{s}_{XY})$ and

$$\mathbf{S}_{XX} = \frac{\mathbf{X}^T(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n)\mathbf{X}}{n-1} \text{ and } \mathbf{s}_{XY} = \frac{\mathbf{X}^T(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n)\mathbf{Y}}{n-1},$$

are sample variance and covariances, respectively. Here, $\mathbf{I}$ is the $n \times n$ identity matrix, $\mathbf{1}$ is the vector of length n, whose elements all equal 1. It follows that, the directions (loadings) for $\mathbf{Y}$ are the right singular vectors of $\mathbf{X}^T\mathbf{Y}$ and loadings for $\mathbf{X}$ are the left singular vectors. The next step is to perform regression of $\mathbf{U}$ on T, namely $\mathbf{U} = \mathrm{T}\boldsymbol{\beta}$. The next column of the projection matrix $\mathbf{P}$ is found by calculating the singular vectors of the residual matrices $(\mathbf{X} - \mathbf{t}\mathbf{p}^T)^T(\mathbf{Y} - \mathrm{T}\boldsymbol{\beta}\mathbf{q}^T)$. The final regression problem is solved for $\mathbf{Y} = \mathbf{UQ} = \mathrm{T}\boldsymbol{\beta}\mathbf{Q} = \mathbf{X}\mathbf{P}^T\boldsymbol{\beta}\mathbf{Q}$, resulting in the PLS estimate:

$$\hat{\mathbf{B}}_{\mathrm{PLS}} = \mathbf{P}^T\boldsymbol{\beta}\mathbf{Q}. \tag{3}$$

PLS transforms $\mathbf{Y}, \mathbf{X} \mapsto \mathbf{U}, \mathrm{T}$ where $\mathbf{U}, \mathrm{T}$ are the scores for $\mathbf{Y}$ and $\mathbf{X}$ respectively. By construction, the loading matrices $\mathbf{P}$ and $\mathbf{Q}$ that correspond to regressions of $\mathbf{Y}$ on $\mathbf{U}$ and $\mathbf{X}$ on T are designed so that $\mathbf{U}$ and T have a maximum correlation such that the prediction rule $\hat{\mathbf{Y}} = \mathbf{Q}\hat{\mathbf{U}} = \mathbf{Q}\mathrm{T}\mathbf{X}$ will have the lowest possible in-sample MSE (mean-squarred error) fit. This contrasts PLS with PCA, which simply looks for variations of maximum explanation in the $\mathbf{X}$ space without regard of the predictive ability of $\mathbf{Y}$. PLS, therefore, provides an optimal pattern matching data transformation methods.

However, as Ehrenberg [1968], N. G. Polson & Scott [2012] argue, there is no logical reason for the output variable to be closely related to the principal components. Moreover, Cook [2007] warns of the pitfalls of using $\mathbf{Y}$ for identifying dimensionality reduction transformations of $\mathbf{X}$, *e.g.* in his analysis of agricultural field trials, the predictions should be

chosen without output (the crop yield), thus making PLS inappropriate. The requirement of $\mathbf{X}$ being stochastic is not always satisfied, *e.g.* in $\mathbf{X}$ arising out of a designed experiment.

PLS has a number of advantages specific to high-dimensional problems, *e.g.* handling multi-collinearity (unlike OLS) as well as multivariate response $\mathbf{Y}$. Note that, when dimension reduction of $\mathbf{X}$ depends on $\mathbf{Y}$, supervised learning reduction is more efficient. The alternative non-linear unsupervised approach is to use an autoencoder. N. Polson et al. [2021] show that one can avoid the linear assumption of PLS by combining with other non-linear techniques and improve the predictive power of a model. Ng [2013] provides a simple summary of PLS and shows that $\mathbf{P}$ are singular vectors of $\mathbf{X}^T\mathbf{Y}$. We defer the additional details of PLS to supplementary section S2.

3.5 Uncertainty Quantification

Our probabilistic model takes the form $\mathbf{Y} \mid F \sim p(\mathbf{Y} \mid F)$, $F = g(\mathbf{BX})$, where $\mathbf{Y}$ is possibly a multivariate output matrix and $\mathbf{X}$ is a $n \times p$ matrix of input variables, and $\mathbf{BX}$ performs dimension reduction. Here $g := g_{\mathbf{W},\mathbf{b}}$ is a deep learner and the parameters $(\hat{\mathbf{W}}, \hat{\mathbf{b}})$ are estimated using traditional SGD methods. The key result, due to Brillinger [2012] and Naik & Tsai [2000] is that $\hat{\mathbf{B}}$ can be estimated consistently, up to a constant of proportionality, using PLS irrespective of the nonlinearity on g. Even though Brillinger [2012] assumes that input $\mathbf{X}$ is Gaussian in order to apply Stein's lemma, this result generalizes to scale-mixtures of Gaussians. See also Iwata [2001] who provides analytical derivation of the uncertainty intervals for ReLU and Probit nonlinear activation functions.

The key insight here is that the lion's share of the UQ can be done at the top layer that outputs $\mathbf{Y}$ as the uncertainty in the dimension reduction of the $\mathbf{X}$ space is much harder to quantify compared to quantifying the uncertainty for the prediction rule. By merging the two cultures, probabilistic model on the first stage and deep learning a subsequent stage, for data transformation of inputs, we can obtain the best of both worlds.

Given a specification of g, the constant of proportionality can also be estimated consistently with $\sqrt{n}$-asymptotics. Hence, to predict at a new level $\mathbf{X}_\star$, we can use the predictive

distribution to make a forecast as well as provide uncertainty bounds.

$$\mathbf{Y}_\star \sim p\left(\mathbf{Y} \mid g_{\hat{\mathbf{W}},\hat{\mathbf{b}}}(\hat{\mathbf{B}}_{\text{PLS}}\mathbf{X})\right)$$
$$\hat{\mathbf{Y}}_* = \mathrm{E}(\mathbf{Y}_* \mid \mathbf{F}_*) = \mathrm{E}_{\mathbf{B}|\mathbf{X},\mathbf{Y}}\left(g(\hat{\mathbf{B}}_{\text{PLS}}\mathbf{X}_*)\right),$$

where $\hat{\mathbf{B}}_{\text{PLS}}$ is given by the left-hand side of (3).

Notice that we can also incorporate uncertainty in the estimation of $\mathbf{B}$ via the posterior $p(\mathbf{B} \mid \mathbf{X}, \mathbf{Y})$. Furthermore, a result of Iwata [2001] can be used to show that the posterior distribution is asymptotically normal. Hence, we can calculate the expectations analytically for activation functions, such as ReLU. As ReLU is convex, Jensen's inequality $g\left(\mathrm{E}(\mathbf{BX})\right) \leq \mathrm{E}\left(g(\mathbf{BX})\right)$ shows that ignoring parameter uncertainty leads to under-prediction.

4 Deep Learning Architectures

Deep learning overcomes many classic drawbacks by *jointly* estimating parameters of both the dimensionality reduction layers and prediction layer based on the full training data $\{\mathbf{X}_i, \mathbf{Y}_i\}_{i=1}^n$, using the information on $\mathbf{Y}$ and $\mathbf{X}$ as well as their relationships, and by using $L \geq 2$ layers. If we choose to use nonlinear layers, we can view a deep learning routine as a hierarchical non-linear factor model, or, more specifically, as a generalized linear model (GLM) with recursively defined nonlinear link functions. Schmidt-Hieber [2020] provides theoretical results for networks with ReLU activation functions, which remains an active area of research. We outline some common DL architectures below.

Feed Forward ReLU: Consider, for example, a simple feed-forward ReLU neural network F. Let $\mathbf{U}$ and T be $n \times L$ matrices, given by

$$\mathrm{T} = \mathbf{X}, \quad \mathbf{Z}_1 = \max(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1, 0),$$
$$\mathbf{U} = \mathbf{Z}_1\mathbf{W}_2 + \mathbf{b}_2, \quad \mathbf{Y} = \mathbf{W}\mathbf{U}.$$

The weights $\mathbf{W}$, $\mathbf{W}_1$ and $\mathbf{W}_2$ are to be learned from training data. We note that sparse Bayesian priors on the weight parameters could be useful to improve the generalization of this, via shrinkage Ghosh et al. [2018].

Convolutional Layer: Partial least squares can also be used as a layer at any stage of deep learning. For example, in a convolutional neural network:

$$\mathbf{Z}_1 = f\left(\sum_{i\in M} \mathbf{X}_i \star \mathbf{W}_i + \mathbf{b}_i\right)$$

where $\mathbf{X}\star\mathbf{W}+\mathbf{b}$ denotes the convolution over the region M, with input image $\mathbf{X}$, weights $\mathbf{W}$ and bias $\mathbf{b}$. Then we can add a PLS layer by regressing the output $\mathbf{Y}$ on $\mathbf{Z}_1$ and performing feature reduction.

Auto-Encoders: An auto-encoder is a deep learning routine which trains the architecture to replicate $\mathbf{X}$ itself, namely $\mathbf{X} = \mathbf{Y}$, via a *bottleneck* structure. This means we select a model $F(\mathbf{X})$ which aims to concentrate the information required to recreate $\mathbf{X}$. Put differently, an auto-encoder creates a more cost-effective representation of $\mathbf{X}$ via a lower dimensional representation. If, for simplicity, we set biases to zero and use one hidden layer ($L = 2$) with only $K < N$ factors, then our *input-output map* becomes

$$Y_j(\mathbf{X}) = F(\mathbf{X})_j = \sum_{k=1}^{K} \mathbf{W}_2^{jk} f\left(\sum_{i=1}^{N} \mathbf{W}_1^{ki}\mathbf{X}_i\right) = \sum_{k=1}^{K} \mathbf{W}_2^{jk}\mathbf{Z}_j, \text{ for } \mathbf{Z}_j = f\left(\sum_{i=1}^{N} \mathbf{W}_1^{ki}\mathbf{X}_i\right),$$

for $j = 1, \ldots, N$, where $f(\cdot)$ is a univariate activation function.

Recurrent Layers: Recurrent neural networks (RNNs) use recurrent layers to capture temporal dependencies with a relatively small number of parameters. RNNs learn temporal dynamics by mapping an input sequence to a hidden state sequence and outputs, via a recurrent layer and a feed-forward layer (*vide* Fig. 2). Let $\mathbf{Y}_t$ denote the observed response and $\mathbf{Z}_t$ are hidden states, then the RNN model is:

$$\begin{aligned} \text{response:} \quad & \hat{Y}_t = f^2(\mathbf{W}_z^2\mathbf{Z}_t + \mathbf{b}^2), \\ \text{hidden states:} \quad & \mathbf{Z}_t = f^1(\mathbf{W}^1[\mathbf{Z}_{t-1}, \mathbf{X}_t] + \mathbf{b}^1), \end{aligned}$$

The time invariant weight matrices $\mathbf{W}^1 = [\mathbf{W}_z^1, \mathbf{W}_x^1]$ and $\mathbf{W}_z^2$ are found through training the network. Here $\mathbf{X}_t$ are extremal inputs up to k lags, $\mathbf{Z}_{t-1}$ are the lagged hidden states, and the hidden state is initialized to zero, $\mathbf{Z}_{t-k} = 0$.

The main difference between RNNs and feed-forward deep learning is the use of a hidden layer with an autoregressive component, here $\mathbf{W}_z^1\mathbf{Z}_{t-1}$. It leads to a network topology in

which each layer represents a time step, indexed by t, in order to highlight the temporal nature. Additional depth can be added to create deep RNNs by stacking layers on top of each other, using the hidden state of the RNN as the input to the next layer. RNNs architectures are learned through the same mechanism described for feedforward architectures. One key difference between implementations of RNNs is that drop-out is not applied to the recurrent connections, only to the non-recurrent connections. In contrast, drop-out is applied to all connections in a given feedforward architecture. We recall here that drop-out in this context refers to a model selection technique designed to avoid over-fitting in deep learning, that removes input dimensions in $\mathbf{X}$ randomly using a Ber(p) indicator and minimizes the mean-squared error while marginalizing over the random node elimination indicator.

Generalizations of RNNs include LSTM (Long short-term memory), which addresses the issue of vanishing or exploding gradients. A memory unit used in LSTM networks allows the network to learn which previous states can be forgotten Hochreiter & Schmidhuber [1997], Schmidhuber & Hochreiter [1997]. Wang & Rocková [2020] directly address the question of uncertainty quantification. Wang et al. [2019] show how to model uncertainty using sampling techniques.

Transformers Transformer is a map $f : \mathbf{X} \mapsto Y$, where both $\mathbf{X}$ and $\mathbf{Y}$ are sequences of length T. Recuurent neural networks or RNNs process one element of a sequence at a time and at every step calculate a lossy summary statistic vector s_t, called the hidden state. Then y_t is predicted from s_t. Transformers, on the other hand, use the entire sequence $\mathbf{X}$ to calculate every element of $\mathbf{Y}$. The key element of a transformer is the attention map [Vaswani et al., 2017] that calculates linear mappings $\mathbf{K} = \mathrm{T}\mathbf{X}$, $\mathbf{V} = \mathbf{U}\mathbf{Y}$, and for new $\mathbf{x}$,

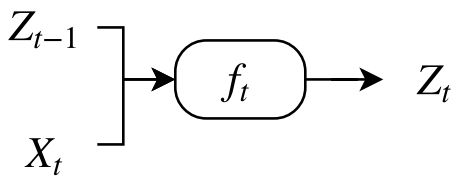


Figure 2: Hidden layer of a Recurrent Neural Network.

calculates $\mathbf{q} = \mathbf{W}\mathbf{x}$ and then predicts new $\mathbf{v}$ as a weighted sum:

$$\hat{\mathbf{v}} = \sum_{i=1}^{n} \alpha_i(q, k_i)v_i,$$

where the attention weights α_i's are calculated using some measure of similarity between q and k_i. For example, they could be calculated using distance-based approaches, such as in traditional non-parametric methods or as an output of a single layer neural network. The seminal paper Bahdanau et al. [2014] introduces the soft-attention mechanism in neural machine translation (NMT) literature that enables the model to dynamically focus on different parts of the source sentence during the translation process, thereby enhancing the model's ability to align and translate input sequences, as well as addressing the limitations of conventional sequence-to-sequence models. The soft attention mechanism leads to a substantial improvements in translation quality and fluency, and is the first explicit occurrence of the attention mechanism. In recent times, transformers have taken precedence over CNNs and RNNs in a substantial portion, if not the majority, of applications. Furthermore, they serve as the foundational architecture behind chatGPT, a substantial language model designed by OpenAI specifically for conversational purposes. See Wolf et al. [2020], Lin et al. [2022] for more details.

5 Merging Deep and Statistical Learning

From the previous sections, the pros and cons of statistical and deep learning architectures are apparent. In this section, we provide a few of these approaches to merge the two frameworks: DL-PLS, PLS-GP, and DL-GP.

5.1 DL-PLS:

Partial least squares algorithm finds projections of the input and output vectors $\mathbf{X} = \mathrm{T}\mathbf{P} + \mathbf{F}$ and $\mathbf{Y} = \mathbf{U}\mathbf{Q} + \boldsymbol{E}$ in such a way that the correlation between the projected input and output is maximized. We propose a new model, called DL-PLS, which introduces nonlinearity $\mathbf{U} = G(\mathrm{T})$ by assuming that $\mathbf{U}$ is a deep learner of T. From Brillinger [2012] it follows that

linear PLS calculates T and $\mathbf{P}$ for arbitrary $G(\cdot)$. We have,

$$\mathbf{Y} = \mathbf{UQ} + \boldsymbol{E}, \quad \mathbf{U} = G(\mathrm{T}), \quad \mathrm{T} = \mathbf{XP}^T,$$

where G is a deep learner. Here $\mathbf{X} = \mathrm{T}\mathbf{P} + \mathbf{F}$ is inverted to $\mathrm{T} = \mathbf{XP}^T$, since $\mathbf{P}^T\mathbf{P} = \mathbf{I}$.

5.2 PLS-GP:

Given a new predictor matrix $\mathbf{X}_* \in \mathbb{R}^{N_\star \times p}$, the same projection $\mathbf{P}$ produces the corresponding $\mathbf{F}_* = [f_{1,*}, ..., f_{L,*}]$. We can use Gaussian process regression to predict $\mathbf{U}_* = [u_{1,*}, ..., u_{L,*}]$ from $\mathbf{F}_*$ as follows,

$$\mathbf{F}_* = \mathbf{X}_*\mathbf{P}^T \text{and } \hat{u}_{k,*} = g_{k,*}(f_{k,*}), k = 1, 2, \ldots, L.$$

where $g_{k,*}$'s are the Gaussian process regression predictors.

$$\begin{bmatrix} u \\ u_* \end{bmatrix} \sim N\left(\begin{bmatrix} g \\ g_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right) \tag{4}$$

where $K = K(t,t) \in \mathbb{R}^{N \times N}$, $K_* = K(t,t_*) \in \mathbb{R}^{N \times N_*}$, and $K_{**} = K(t_*,t_*) \in \mathbb{R}^{N_* \times N_*}$, and $K(\cdot,\cdot)$ is a kernel function. The conditional mean, g_*, is:

$$g_*(t_*) = g(t_*) + K_*^T K^{-1}(u - g(t)).$$

Then the prediction of Y is: $\hat{\mathbf{Y}}_* = \hat{\mathbf{U}}_*\, Q$. The key to both DL-PLS and PLS-GP is that the deep learner is viewed as a non-linear transformation rule, via feature expansion and reduction, as specified earlier. The uncertainty is introduced in connecting this transformed input to the final output Y. Clearly, this approach does not attempt to model the uncertainty at the lower layers, for example, in the network weights. However, we argue that it is the top level uncertainty that is the most crucial, as well as the most tractable, in order to achieve predictive uncertainty quantification, which traditional DL models lack. We proceed to provide numerical evidence for the benefits of these proposed approaches.

5.3 DL-GP

Both PLS and DL learn a low dimensional representation of the input vector. We build a combined DL-GP model. The goal of the DL model is to find ϕ, which is a reduced

dimensionality representation of the input vector $\mathbf{X}$. Then to use a Gaussian Process to model relations between low dimensional inputs ψ and the output $\mathbf{Y}$. Our architecture is shown in Fig. 3.

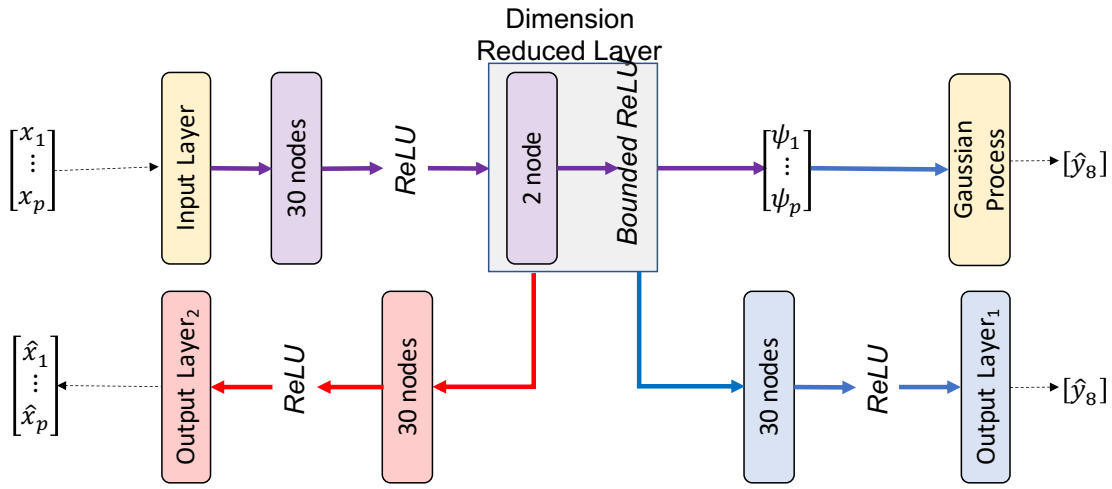
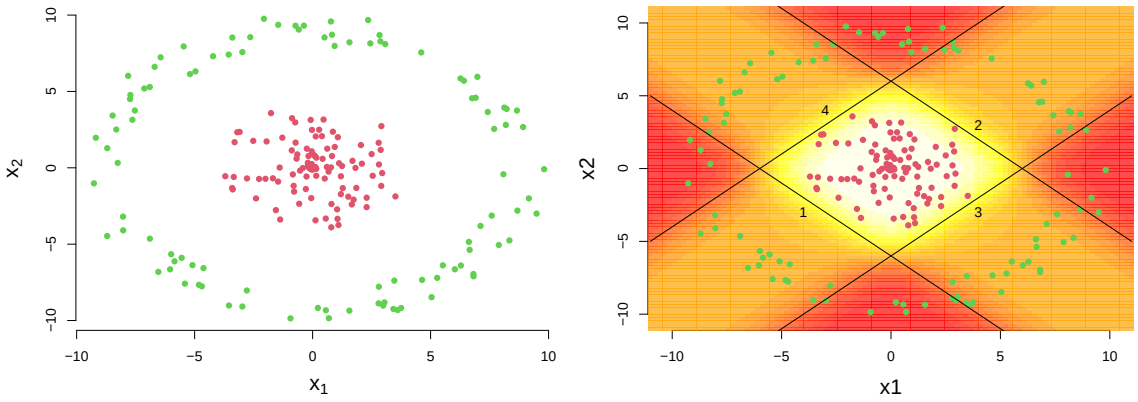


Figure 3: Architecture of the DL-GP model

6 Applications

In this section we provide two stylized examples to demonstrate the ability of deep learning models to discover low dimensional structures in the data. The third example shows how Gaussian Process model can be combined with a deep learning for dimensionality reduction and provide uncertainty quantification. First, we demonstrate the similarity between a tree model and a one-layer neural network model by showing how they classify the observations forming a simulated doughnut dataset shown in Figure 4(a). The data has two-dimensional input x_1 and x_2 and a one dimensional binary output, color coded green/red. It is clear that there is no single line that separates green and red observations. Thus, the linear separator does not work in this case. However, we can use four lines to split the data into nine regions, as shown. Figure 4(b) shows the four-line model and colors each of the region according to probability of a point inside this region being a green point. The red color corresponds to probability close to 1 and white color to probability close to 0. Given this split, it is easy to design a predictive rule for classifying the red and green dots. If a point is on the left of lines



(a) Simulated dataset

(b) Hyperplanes and probability heat map for classification based on the four hyperplanes

Figure 4: Training data set and four hyper-planes defined by a hidden layer of a neural network

2, 3 and on the right of lines 4, 1, then classify as red and classify as green otherwise. The four lines are given by a linear system $a = Bx$, $Z = \sigma(a)$. Finally, we perform classification by a logistic regression:

$$\mu = -3 + Z_1 + Z_2 + Z_3 + Z_4$$
$$P(Y = 1 \mid X) = e^{\mu}/(1 + e^{\mu}).$$

Figure 4(b) shows the heatmap of the $P(Y = 1 \mid X)$ with red being 1 and white 0. We can also visualize the deep learning model using a tree-like diagram to highlight the similarity between the two approaches. To illustrate this, consider the well-known doughnut problem in Fig. 5. Among the several solutions, the simplest is to find the input space data augmentation $\phi(X_1, X_2) \mapsto (X_1^2, X_2^2, \sqrt{2}X_1X_2)$, and find one hyperplane that separates the data in this new coordinate system.

6.1 Identifying Nonlinear Systems

To illustrate the merging of deep learning and PLS, we start with multivariate observation and state an identification problem. We start with a simple example to show how sequential steps of PLS are used to estimate the coefficient matrix P. We simulate the nonlinear data set using the generative model $\mathbf{Y} = |10 + \mathbf{XP}| + \epsilon$ where $\mathbf{X} \in \mathbb{R}^{N \times 100}$ and $\mathbf{P} \in \mathbb{R}^{100 \times 1}$. Now

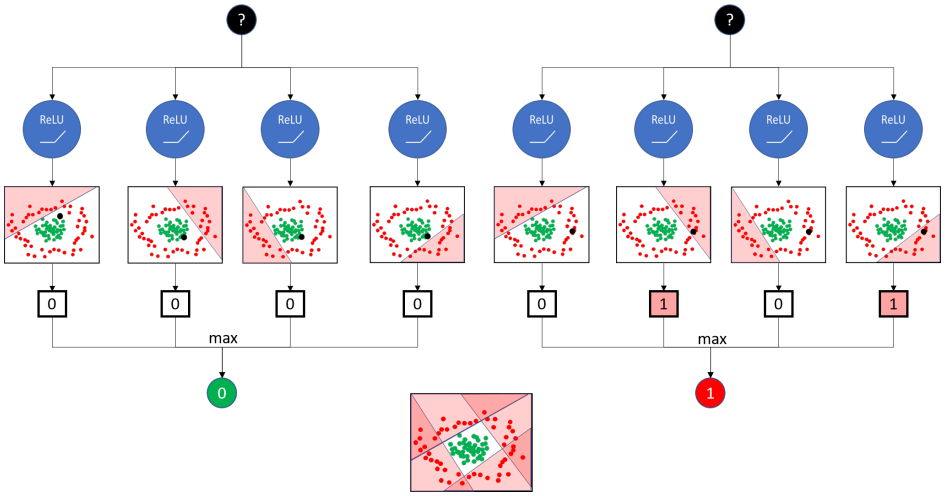


Figure 5: Deep ReLU network

we use PLS and OLS and OLS to estimate the matrix $\mathbf{P}$ and to calculate $\hat{\mathbf{Y}} = \mathbf{X}\hat{\mathbf{P}}$. In the PLS case, we use $\hat{\mathbf{Y}} = \mathbf{X}\hat{\mathbf{P}}$. We perform estimation assuming IID normal noise terms. Since it is a simple linear system, the OLS directly identifies $\mathbf{P}$. However, it is instructive to see how the sequential PLS performs the same task. Figure 6 shows the results of the estimation for both OLS and PLS and compare the predicted output with the truth $\mathbf{Y}$. We can see

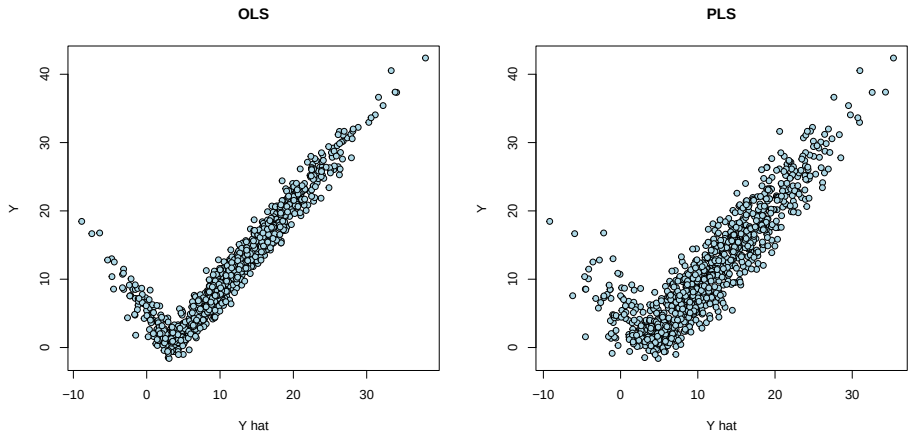
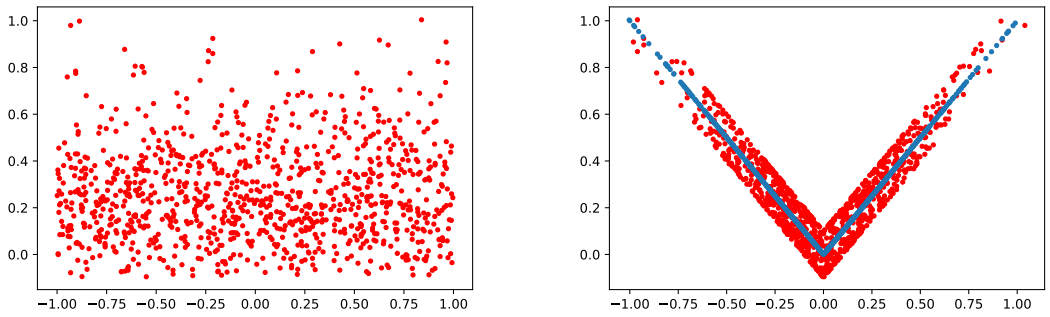


Figure 6: OLS and PLS estimators recover the nonlinear absolute function.

that, both OLS or PLS can identify the system.

6.2 Simulated Example

First, we consider how deep learning can be used as a sufficient dimensionality reduction technique to identify lower dimensional features that can be later used as inputs to statistical models. We start with a synthetic example that demonstrates an application of one layer neural network to find a piece-wise linear structure in the data. Consider real-valued function $F(\mathbf{X}) = \|u^T\mathbf{X}\|$ where $\mathbf{X} = (X_0, \ldots, X_{100})^T$. We generate input-output pairs $\{\mathbf{X}_i, Y_i\}_{i=1}^n$, where $\mathbf{X}_{ij} \sim \mathcal{U}[-1, 1]$, and $Y_i = F(\mathbf{X}_i)+\epsilon$, $\epsilon \sim \mathcal{N}(0, 0.01)$. There is one-dimensional structure in the input-output relations, which is represented by a ridge function $\|u^T\mathbf{X}\|$. We use neural network to identify this one dimensional structure, and we introduce a one-dimensional bottleneck in our neural network. The overall architecture is as follows $\hat{Y} = f(\phi(\mathbf{X}))$, where $\phi : \mathbb{R}^p \mapsto \mathbb{R}$, and $f : \mathbb{R} \mapsto \mathbb{R}$. Both ϕ and f are single layer neural networks.



(a) X_0 vs $f(X)$ **(b)** $\phi(X)$ vs $f(X)$

Figure 7: Piece-wise linear structure via $f(\cdot)$ and $\phi(\cdot)$

6.3 MARTHE Dataset

The MARTHE dataset Surjanovic & Bingham [n.d.] consists of 300 input-output pairs, where outputs are realizations of the numerical simulation of Strontium-90 transport in the upper aquifer of the RRC "Kurchatov Institute" radwaste disposal site in Moscow, Russia Volkova et al. [2008]. The input vector has 20 components describing physical properties of the aquifer, such as hydraulic conductivity, porosity and transversal dispersivity. The outputs are contaminant concentrations at 10 wells, and we use the output at one of them: well

number 8 to compare the merged architectures: DL-GP and PLS-GP.

Gaussian Processes (GPs) have been used to provide an output map together with uncertainty bands. We use a zero mean GP with separable anisotropic Gaussian covariance function which is the sum of inverse exponentiated squared difference plus the nugget:

$$C_d(X, X') = \exp\left\{-\sum_{i=1}^{p}(X_i - X_i^{'})^2/d_i\right\} + g\delta_{x,X'}$$

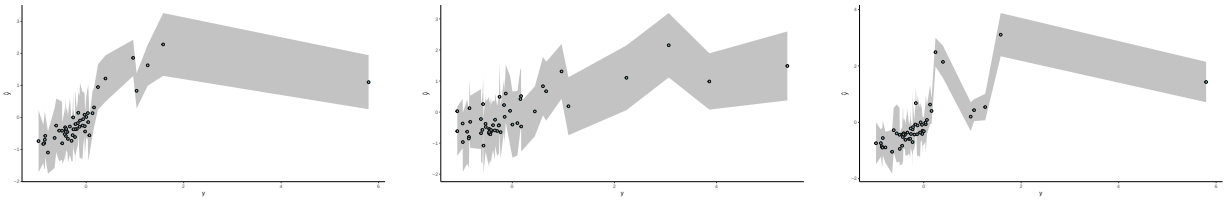
The length-scale parameters of the covariance function $d_1, \ldots, d_p$ and the nugget parameter g were estimated by maximizing the Bayesian integrated log likelihood Gramacy [2016, 2020]. Figure 8 shows the scatter plot of the predicted values $\hat{Y}$ versus the actual observations Y, along with 95% credible interval from the GP model and its competitors: PLS-GP and DL-GP, introduced in section 5.

Our merging of two cultures leads to an improved fit and predictive map as follows. First, we use partial least squares (PLS) N. Polson et al. [2021] to provide feature selection. Remember that the learned features depend on both the input and output pairs, leading to an optimal mean squared error in-sample fit. Moreover, PLS provides a dimension reduction that helps in the GP model at the top level. Dimensionality reduction is crucial in many applications, e.g. in sequential design of experiment, when lower dimensionality allows for more efficient exportation of the input space MacKay [1992], Fedorov & Hackl [1997]. We can view the 'merged cultures' model as a hierarchical latent feature model, where the stochastic GP model is used to provide a predictive distribution at the top level.

Figure 8 shows the predicted and actual scatter plots, and Table 1 compares the mean absolute prediction error (MAPE) and root-mean-square error (RMSE) for the three candidates: plain GP, PLS+GP and DL+GP, and shows that the DL+GP outperforms the competitors in terms of both these metrics.

Table 1: Out-of-sample performance of candidate models for MARTHE data.

	Plain GP	PLS-GP	DL-GP
RMSE	4.5	1.60	0.89
MAPE	0.8	0.73	0.16



(a) PLS-GP (b) DL-GP (c) Vanilla GP

Figure 8: Out-of-sample predictions from plain GP model and GP model that uses PLS scores versus DL scores as inputs. 10 DL components and 14 PLS components were used.

7 Discussion

One goal of statistics is to build predictive models along with uncertainty and to develop an understanding of the data generating mechanism. Data models are well studied in statistical literature, but often do not provide enough flexibility to learn the input-output relations. Closed box predictive rules, such as trees and neural networks, are more flexible learners but do not provide predictive uncertainties or the ability for probabilistic modeling.

Data can be thought of as generated by a closed box mechanism, on which a vector of input variables $\mathbf{X}$ is mapped to an output (or response vector Y). One goal is prediction, to be able to assign a response variable Y_* to a new (unseen before) input $\mathbf{X}_*$. Two cultures have emerged for performing this task: stochastic methods with parameters or closed box predictions rules. The questions we are concerned with, broadly speaking, are: *What makes a good statistical model?* and, *What makes a good prediction rule?* Given a statistical model (and computation) there are theoretical tools leading to an optimal prediction rule. However, in high-dimensional problems finding good models is challenging, and this is where closed box methods shine. One also needs a good prior distribution that gets updated in the light of evidence. Our methodology provides a merging of these two cultures. We show that deterministic closed box rule can be used as a transformation of high dimensional input and outputs. Hidden features lie on the transformed space, and are empirically learned as opposed to theoretically specified. Statistical modeling then provides uncertainty assessment via traditional Bayesian updating methods, complementing closed box approaches.

Supplementary Material

The supplementary material contains further technical descriptions on prediction via shrinkage and ensemble, Brillinger estimation, and additional figures.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., & Bellemare, M. (2021). Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, *34*.

Alaa, A. M., & van der Schaar, M. (2019). Attentive State-Space Modeling of Disease Progression. *Advances in Neural Information Processing Systems*, *32*, 11338–11348.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Binois, M., & Wycoff, N. (2021). A survey on high-dimensional gaussian process modeling with application to Bayesian optimization. *arXiv preprint arXiv:2111.05040*.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., & Kasneci, G. (2021). *Deep Neural Networks and Tabular Data: A Survey.*

Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, *16*(3), 199–231.

Brillinger, D. R. (2012). A Generalized Linear Model With "Gaussian" Regressor Variables. In P. Guttorp & D. Brillinger (Eds.), *Selected Works of David Brillinger* (pp. 589–606). New York, NY: Springer.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Chun, H., & Keleɛs, S. (2010). Sparse partial least squares regression for simultaneous dimension reduction and variable selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *72*(1), 3–25.

Cook, R. D. (2007). Fisher lecture: Dimension reduction in regression. *Statistical Science*, *22*(1), 1–26.

Cook, R. D., & Forzani, L. (2009). Likelihood-based sufficient dimension reduction. *Journal of the American Statistical Association*, *104*(485), 197–208.

Cook, R. D., & Forzani, L. (2018). Big data and partial least-squares prediction. *Canadian Journal of Statistics*, *46*(1), 62-78. doi: https://doi.org/10.1002/cjs.11316

Cutler, A., Cutler, D. R., & Stevens, J. R. (2008). Tree-based methods. In *High-dimensional data analysis in cancer research* (pp. 1–19). Springer.

Damianou, A., & Lawrence, N. D. (2013). Deep gaussian processes. In *Artificial intelligence and statistics* (pp. 207–215).

Dean, J. (2020). 1.1 the deep learning revolution and its implications for computer architecture and chip design. In *2020 ieee international solid-state circuits conference-(isscc)* (pp. 8–14).

Delaigle, A., & Hall, P. (2012). Methodology and theory for partial least squares applied to functional data. *The Annals of Statistics*, *40*(1), 322–352.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dixon, M. F., Polson, N. G., & Sokolov, V. O. (2019). Deep learning for spatio-temporal modeling: Dynamic traffic flows and high frequency trading. *Applied Stochastic Models in Business and Industry*, *35*(3), 788–807.

Ehrenberg, A. S. C. (1968). The elements of lawlike relationships. *Journal of the Royal Statistical Society. Series A (General)*, *131*(3), 280–302.

Esteva, A., Chou, K., Yeung, S., Naik, N., Madani, A., Mottaghi, A., ... Socher, R. (2021). Deep learning-enabled medical computer vision. *NPJ digital medicine*, *4*(1), 1–9.

Fan, J., Ma, C., & Zhong, Y. (2021). A selective overview of deep learning. *Statistical Science*, *36*(2), 264–290.

Fedorov, V. V., & Hackl, P. (1997). *Model-oriented design of experiments* (Vol. 125). Springer Science & Business Media.

Fei, J., Zhao, J., Sun, S., & Liu, Y. (2018). Active learning methods with deep Gaussian processes. In *Neural information processing: 25th international conference, iconip 2018, siem reap, cambodia, december 13–16, 2018, proceedings, part iii 25* (pp. 473–483).

Frank, I. E., & Friedman, J. H. (1993). A Statistical View of Some Chemometrics Regression Tools. *Technometrics*, *35*(2), 109–135.

Ghosh, S., Yao, J., & Doshi-Velez, F. (2018). Structured variational learning of Bayesian neural networks with horseshoe priors. In *International conference on machine learning* (pp. 1744–1753).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Gramacy, R. B. (2016). lagp: large-scale spatial modeling via local approximate gaussian processes in r. *Journal of Statistical Software*, *72*(1), 1–46.

Gramacy, R. B. (2020). *Surrogates: Gaussian process modeling, design and optimization for the applied sciences*. Boca Raton, Florida: Chapman Hall/CRC. (http://bobby.gramacy.com/surrogates/)

Gramacy, R. B., & Lee, H. K. H. (2008). Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, *103*(483), 1119–1130.

Gu, S., Kelly, B., & Xiu, D. (2021). Autoencoder asset pricing models. *Journal of Econometrics*, *222*(1), 429–450.

Harding, E. F. (1967). The number of partitions of a set of n points in k dimensions induced by hyperplanes. *Proceedings of the Edinburgh mathematical society*, *15*(4), 285–289.

Hastie, T., & Tibshirani, R. (2009). *& friedman, j.(2008). the elements of statistical learning; data mining, inference and prediction.* Springer, New York.

Heaton, J., Polson, N., & Witte, J. H. (2017). Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, *33*(1), 3–12.

Helland, I. S. (1990). Partial Least Squares Regression and Statistical Models. *Scandinavian Journal of Statistics*, *17*(2), 97–114.

Higdon, D., Gattiker, J., Williams, B., & Rightley, M. (2008). Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, *103*(482), 570–583.

Hochreiter, S., & Schmidhuber, J. (1997). LSTM can solve hard long time lag problems. In *Advances in neural information processing systems* (pp. 473–479).

Huber, P. J. (1985). Projection pursuit. *The Annals of Statistics*, 435–475.

Iwata, S. (2001, August). Recentered and Rescaled Instrumental Variable Estimation of Tobit and Probit Models with Errors in Variables. *Econometric Reviews*, *20*(3), 319–335.

Jacquier, P., Abdedou, A., Delmas, V., & Soulaïmani, A. (2021). Non-intrusive reduced-order modeling using uncertainty-aware deep neural networks and proper orthogonal decomposition: Application to flood modeling. *Journal of Computational Physics*, *424*, 109854.

Li, K.-C. (1991). Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, *86*(414), 316–327.

Li, L., Qin, L., Xu, Z., Yin, Y., Wang, X., Kong, B., ... others (2020). Artificial intelligence distinguishes COVID-19 from community acquired pneumonia on chest CT. *Radiology*.

Lim, B., Arık, S. , Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, *37*(4), 1748-1764.

Lin, T., Wang, Y., Liu, X., & Qiu, X. (2022). A survey of transformers. *AI Open*.

Lindley, D. V. (1968). The choice of variables in multiple regression. *Journal of the Royal Statistical Society. Series B (Methodological)*, *30*(1), 31–66.

MacKay, D. J. (1992). Information-based objective functions for active data selection. *Neural computation*, *4*(4), 590–604.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, *36*(1), 54–74.

Malthouse, E. C., Tamhane, A. C., & Mah, R. (1997). Nonlinear partial least squares. *Computers & Chemical Engineering*, *21*(8), 875–890.

Massy, W. F. (1965). Principal components regression in exploratory statistical research. *Journal of the American Statistical Association*, *60*(309), 234–256.

McCullagh, P. (2019). *Generalized linear models* (2nd ed.). London: Chapman and Hall.

McDermott, P. L., & Wikle, C. K. (2019). Bayesian recurrent neural network models for forecasting and quantifying uncertainty in spatial-temporal data. *Entropy*, *21*(2), 184.

Naik, P., & Tsai, C.-L. (2000). Partial least squares estimator for single-index models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *62*(4), 763-771.

Ng, K. S. (2013). A simple explanation of partial least squares. *The Australian National University, Canberra*, 1–10.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*(6583), 607.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., & Liao, Q. (2017). Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, *14*(5), 503–519.

Polson, N., Sokolov, V., & Xu, J. (2021). Deep Learning Partial Least Squares. *arXiv preprint arXiv:2106.14085*.

Polson, N. G., & Scott, J. G. (2012). Local shrinkage rules, Lévy processes and regularized regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *74*(2), 287-311.

Polson, N. G., & Sokolov, V. O. (2017). Deep learning for short-term traffic flow prediction. *Transportation Research Part C: Emerging Technologies*, *79*, 1–17.

Popov, S., Morozov, S., & Babenko, A. (2019). Neural oblivious decision ensembles for deep learning on tabular data.

Rigter, M., Lacerda, B., & Hawes, N. (2021). Risk-Averse Bayes-Adaptive Reinforcement Learning. *Advances in Neural Information Processing Systems*, *34*, 1142–1154.

Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge university press.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International conference on medical image computing and computer-assisted intervention* (pp. 234–241).

Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (1995). Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, 1–34.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191.

Schmidhuber, J., & Hochreiter, S. (1997). Long Short-Term Memory. *Neural Comput*, *9*(8), 1735–1780.

Schmidt-Hieber, J. (2020). Nonparametric regression using deep neural networks with relu activation function. *The Annals of Statistics*, *48*(4), 1875–1897.

Shallue, C. J., & Vanderburg, A. (2018, Jan). Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90. *The Astronomical Journal*, *155*(2), 94.

Surjanovic, S., & Bingham, D. (n.d.). *Virtual library of simulation experiments: Test functions and datasets.* Retrieved December 20, 2019, from http://www.sfu.ca/~ssurjano.

Sutskever, I., & Hinton, G. E. (2008). Deep, narrow sigmoid belief networks are universal approximators. *Neural computation*, *20*(11), 2629–2636.

Tran, M.-N., Nguyen, N., Nott, D., & Kohn, R. (2019). Bayesian deep net glm and glmm. *Journal of Computational and Graphical Statistics*, 1–17.

Triebe, O., Hewamalage, H., Pilyugina, P., Laptev, N., Bergmeir, C., & Rajagopal, R. (2021). *NeuralProphet: Explainable Forecasting at Scale.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Volkova, E., Iooss, B., & Van Dorpe, F. (2008). Global sensitivity analysis for a numerical model of radionuclide migration from the rrc "Kurchatov Institute" radwaste disposal site. *Stochastic Environmental Research and Risk Assessment*, *22*(1), 17–31.

Wang, Y., Polson, N. G., & Sokolov, V. O. (2019). Scalable data augmentation for deep learning. *arXiv preprint arXiv:1903.09668*.

Wang, Y., & Rocková, V. (2020). Uncertainty quantification for sparse deep learning. In *International conference on artificial intelligence and statistics* (pp. 298–308).

Wold, H. (1975). Soft modelling by latent variables: the non-linear iterative partial least squares (nipals) approach. *Journal of Applied Probability*, *12*(S1), 117–142.

Wold, H. (1985). Partial Least Squares. In *Kotz, S., Johnson, N.L. (Eds.), Encyclopedia of the Statistical Sciences* (Vol. 6, pp. 581–591). New York: Wiley.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... others (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38–45).