Matrix Multilayer Perceptron

Anonymous authors
Paper under double-blind review

Abstract

Models that output a vector of responses given some inputs, in the form of a conditional mean vector, are at the core of machine learning. This includes neural networks such as the multilayer perceptron (MLP). However, models that output a symmetric positive definite (SPD) matrix of responses given inputs, in the form of a conditional covariance function, are far less studied, especially within the context of neural networks. Here, we introduce a new variant of the MLP, referred to as the matrix MLP, that is specialized at learning SPD matrices. Our construction not only respects the SPD constraint, but also makes explicit use of it. This translates into a model which effectively performs the task of SPD matrix learning even in scenarios where data are scarce. We present an application of the model in heteroscedastic multivariate regression, including convincing performance on six real-world datasets.

1 Introduction

For certain applications, it is desirable to construct a conditional covariance matrix as a function of the input (the explanatory variable). The problem arises, for instance, in spatial (and spatio-temporal) statistics, in relation to the heteroscedastic multivariate regression (e.g., [Pourahmadi, 1999], [Hoff & Niu, 2012], [Fox & Dunson, 2015]), where we deal with multivariate response measurements for which the typical assumption of homoscedasticity may not be suitable. In such cases, we require models that estimate the covariance matrix that captures the spatial variations in correlations between the elements of the response vector. The covariance matrix is a symmetric positive definite (SPD) matrix which can be challenging to estimate due to its non-Euclidean geometry ([Pennec et al., 2005]). The central problem that this work is concerned with is learning SPD matrices using neural networks.

To motivate our discussion, consider solving the problem of SPD matrix learning using a multilayer perceptron (MLP) as an example of a fully connected neural network (e.g., [Goodfellow et al., 2016]). To meet the SPD constraint, one would need to tailor the output layer of the MLP so that the estimated covariance matrix satisfies the SPD requirement. One possible approach would be to use the Cholesky decomposition. The main concern is that this approach does not take into account the non-Euclidean geometry of the SPD matrices. Using empirical evaluations, we will show that the use of “wrong geometry” results in the poor estimation of the SPD matrices in particular where data are scarce.

The primary objective here is to design a nonlinear architecture using neural networks that can effectively perform the task of SPD matrix learning. More specifically, our main contribution is to show how to alter the architecture of the MLP in such a way that it not only respects the SPD constraint, but also makes an explicit use of it. We will achieve this by: 1) Explicitly taking the non-Euclidean geometry of the underlying SPD manifolds (e.g., [Pennec et al., 2005]) into account by designing a new loss function, and 2) by deriving a new backpropagation algorithm ([Rumelhart et al., 1986]) that respects the SPD nature of the matrices. This new model will be referred to as the matrix multilayer perceptron (mMLP). The mMLP makes use of positive-definite kernels to satisfy the SPD requirement across all layers. Hence, it provides a natural way of enabling deep SPD matrix learning.

We take a step-by-step approach in the development of the model. We first develop a simplified version of the resulting model that is designed for learning SPD matrices (Section 3). We then extend this model into its most general form that can be used for joint estimation of the conditional mean function and the conditional covariance function in a mean-covariance regression setting (Section 3.2). An application of the model is discussed in the context of the heteroscedastic multivariate regression.

1 An implementation of this work will be made available via GitHub.
2 RELATED WORK

SPD manifold metric. Earlier approaches for analyzing SPD matrices relied on the Euclidean space. But over the past decade, several studies suggest that non-Euclidean geometries such as the Riemannian structure may be better suited (e.g., [Arsigny et al., 2006; Pennec et al., 2005]). In this work, we consider the von Neumann divergence (e.g., Nielsen & Chuang, 2000) as our choice of the SPD manifold metric which is related to the Riemannian geometry. Previously, [Tsuda et al., 2005] used this divergence in derivation of the matrix exponentiated gradients. Their work suggests its effectiveness for measuring dissimilarities between positive definite (PD) matrices.

SPD manifold learning. There are multiple approaches towards the SPD matrix learning, via the flattening of SPD manifolds through tangent space approximations (e.g., [Oncel Tuzel, 2008; Fathy et al., 2016], mapping them into reproducing kernel Hilbert spaces (Harandi et al., 2012; Minh et al., 2014), or geometry-aware SPD matrix learning (Harandi et al., 2014). While these methods typically employ shallow learning, the more recent line of research aims to design a deep architecture to non-linearly learn target SPD matrices ([Ionescu et al., 2015; Huang & Gool, 2017; Masci et al., 2015; Huang et al., 2018]). Our method falls in this category but differs in the problem formulation. While the previous methods address the problem where the input is an SPD matrix and the output is a vector, we consider the reverse problem where the input is a matrix with an arbitrary size and the output is an SPD matrix.

Backpropagation. Our extension of the matrix backpropagation differs from the one introduced by [Ionescu et al., 2015]. In their work, the necessary partial derivatives are computed using a two-step procedure consisting of first computing the functional that describes the variations of the upper layer variables with respect to the variations of the lower layer variables, and then computing the partial derivatives with respect to the lower layer variables using properties of the matrix inner product. In contrast, we make use of the concept of \( \alpha \)-derivatives (Magnus, 2010) and its favorable generalization properties to derive a procedure which closely mimics the standard backpropagation.

3 MATRIX MULTILAYER PERCEPTRON

Preliminaries: Matrix \( \alpha \)-derivative. Throughout this work we adopt the narrow definition of the matrix derivatives known as the \( \alpha \)-derivative (Magnus, 2010) in favor of the broad definition, the \( \omega \)-derivative. The reason for this is that the \( \alpha \)-derivative has better generalization properties. This choice turned out to be crucial in the derivation of the mMLP’s backpropagation routine which involves derivatives of matrix functions w.r.t. the matrix of variables.

Definition: Let \( F \) be an \( m \times n \) matrix function of an \( n \times q \) matrix of variables \( X \). The \( \alpha \)-derivative of \( F(X) \) is defined as (Magnus, 2010) Definition 2)

\[
D_X F := \frac{\partial \text{vec} F(X)}{\partial (\text{vec} X)} ,
\]

where \( D_X F \) is an \( mp \times nq \) matrix which contains all the partial derivatives such that each row contains the partial derivatives of one function with respect to all variables, and each column contains the partial derivatives of all functions with respect to one variable.

For convenience, the \( \alpha \)-derivative’ basic properties, including the product rule and the chain rule, are summarized in Appendix B.

3.1 THE BASIC FORM OF THE mMLP

Activation matrix function. Let \( Z = (z_1, \ldots, z_d) \) denote a matrix of variables \( z_i \in \mathbb{R}^d \). The activation function \( K(Z) \) defines a matrix function in the form of \( [K(Z)]_{i,j} = \kappa(z_i, z_j) \), \( \forall i,j \in \{1, \ldots, d\} \), where \( \kappa \) is some differentiable activation function outputting scalar values. In the following, we restrict ourselves to the kernel functions which form PD activation matrix functions. For numerical stability reasons, irrespective of the functional form of \( \kappa \), we normalize the resulting matrix. This can be achieved by enforcing the trace-one constraint, \( H(Z) = K(Z)/\text{tr}(K(Z)) \), where \( H \) denotes a differentiable SPD activation matrix function of trace one. Without loss of generality, throughout this work, we use the Mercer sigmoid kernel (Carrington et al., 2014) defined as

\[
\kappa(z_i, z_j) = \tanh(\alpha z_i + \beta) \odot \tanh(\alpha z_j + \beta),
\]

Under review as a conference paper at ICLR 2020
where $\alpha$ and $\beta$ denote the slope and the intercept, respectively. Furthermore, \odot denotes the dot product. In all experiments, we use default values of $\alpha = 1$ and $\beta = 0$. The $\alpha$-derivative of the Mercer sigmoid kernel is computed in Appendix [E].

**Model construction.** Let $X \in \mathbb{R}^{p_1 \times p_2}$ indicate the input matrix and $Y \in \mathbb{R}^{d_0 \times d_0}$ indicate the corresponding output matrix, an SPD matrix of trace one. The mMLP of $L$ hidden layers is shown as $mMLP: X \rightarrow \hat{Y}$ and constructed as

\[
\begin{align*}
\hat{Y} &= \mathcal{H}(Z_0), \\
H_i &= \mathcal{H}(Z_i), \\
H_{L+1} &= \mathcal{H}(Z_{L+1}),
\end{align*}
\]

\[
\begin{align*}
Z_0 &= W_0 H_1 W_0^\top + B_0, \\
Z_l &= W_l H_{l+1} W_l^\top + B_l, \quad \forall 1 \leq l \leq L, \\
Z_{L+1} &= W_{L+1} \text{vec}(W_{L+1} 1_{p_1 p_2})^\top + B_{L+1}.
\end{align*}
\]

The pair of $W_l \in \mathbb{R}^{d_l \times d_{l+1}}, \forall 0 \leq l \leq L$, and $W_{L+1} \in \mathbb{R}^{d_{L+1} \times p_2}$ are the weight matrices, $B_l \in \mathbb{R}^{d_l \times d_{l}}, \forall 0 \leq l \leq L+1$, are the bias matrices, $Z_l \in \mathbb{R}^{d_l \times d_l}, \forall 0 \leq l \leq L+1$, are the latent input matrices, and $H_l \in \mathbb{R}^{d_l \times d_l}, \forall 1 \leq l \leq L + 1$, are latent output SPD matrices of trace one.

**Design choice.** In the construction of [3], we have ensured that $H_l$ are SPD matrices of trace one across all layers as opposed to only at the output layer. The idea behind this design choice is to propagate the nonlinearities introduced via the SPD activation matrix functions through all layers. This design choice turned out to be more effective than the alternative, and arguably simpler, design where the SPD requirement is met only at the output layer. We will discuss this further in Section [5.1.3] where we also present an illustrative numerical example.

**Loss function.** We consider the normalized von Neumann divergence (e.g., Nielsen & Chuang 2000), also commonly known as the quantum relative entropy (QRE), as the base for the loss function. For two arbitrary SPD matrices of trace one, $\Phi$ and $\tilde{\Phi}$, the normalized von Neumann divergence is defined as:

\[
\Delta_{\text{QRE}}(\hat{\Phi} \mid \Phi) = \text{tr}(\hat{\Phi} \log \hat{\Phi} - \Phi \log \Phi),
\]

where $\log$ denotes the matrix logarithm (taking $\Phi$ as an example, it is computed using $\log \Phi = V \text{diag}(\log \lambda)V^\top$, where $V$ and $\lambda$ are the matrix of eigenvectors and the vector of eigenvalues from the eigendecomposition of $\Phi$). The von Neumann divergence is asymmetric. However, it can be symmetrized by using the fact that the von Neumann entropy of trace one follows the class of generalized quadratic distances (Nielsen & Nock 2007). Hence, we define the loss function $\ell_{\text{QRE}}$:

\[
\ell_{\text{QRE}}(\hat{Y}, Y) = \frac{1}{2}(\Delta_{\text{QRE}}(\hat{Y} \mid Y) + \Delta_{\text{QRE}}(Y \mid \hat{Y})).
\]

Taking $\alpha$-derivative of $\ell_{\text{QRE}}$ involves taking partial derivatives through the eigendecomposition. In the following, we derive a method for analytically computing the derivative of $\ell_{\text{QRE}}$.

**The $\alpha$-derivative of the symmetrized von Neumann divergence.** For the loss function defined in [5], using the $\alpha$-derivative’s product rule and chain rule (refer to Appendix [E]), we obtain

\[
D_{\psi} \ell = \frac{1}{2} D_{\psi} \text{tr}((\hat{Y} - Y) \log \hat{Y}) - \frac{1}{2} D_{\psi} \text{tr}(\hat{Y} \log Y),
\]

where the above two terms are computed using

\[
\begin{align*}
D_{\psi} \text{tr}((\hat{Y} - Y) \log \hat{Y}) &= \left(\text{vec}(\log \hat{Y})^\top \right)^\top + \left(\begin{array}{c}
\text{vec}(\hat{Y}^\top - Y^\top) \odot \frac{\partial}{\partial \hat{Y}^{	op}} \log \hat{Y} \\
\text{vec}(\hat{Y}^\top - Y^\top) \odot \frac{\partial}{\partial Y^{	op}} \log \hat{Y} \\
\vdots \\
\text{vec}(\hat{Y}^\top - Y^\top) \odot \frac{\partial}{\partial Y_{d_0 d_0}} \log \hat{Y}
\end{array}\right)^\top, \\
D_{\psi} \text{tr}(\hat{Y} \log Y) &= \left(\text{vec}(\log Y)^\top \right)^\top.
\end{align*}
\]

\footnote{Note that there are multiple ways of symmetrizing the von Neumann divergence. Our choice in [5] resembles to the $\alpha$-Jensen–Shannon divergence for $\alpha = 1$ (Nielsen [2010]).}
The remaining part in the computation of (6) is to evaluate $\frac{\partial}{\partial Y_{ij}} \log \hat{Y}$, for all $i, j \in \{1, \ldots, d_0\}$, which involves taking derivatives through the eigendecomposition. In the following, we take a similar approach as in Papadopoulo & Lourakis (2000) to compute the necessary partial derivatives.

Let $\hat{Y} = Y \text{diag}(\lambda_1, \ldots, \lambda_{d_0})^T$ be the eigendecomposition. We can write

$$\frac{\partial}{\partial Y_{ij}} \log \hat{Y} = \frac{\partial}{\partial Y_{ij}} Y \Lambda Y^T,$$

where $\Lambda = \text{diag}(\log \lambda_1, \ldots, \log \lambda_{d_0})$.

By multiplying (9) from left and right by $Y^T$ and $Y$ respectively, we obtain:

$$Y^T \frac{\partial}{\partial Y_{ij}} \log \hat{Y} Y = Y^T \frac{\partial}{\partial Y_{ij}} \Lambda + \frac{\partial \Lambda}{\partial Y_{ij}} + \Lambda \frac{\partial Y^T}{\partial Y_{ij}} Y = \Xi_{ij}(Y) \Lambda + \frac{\partial \Lambda}{\partial Y_{ij}} - \Lambda \Xi_{ij}(Y), \quad (10)$$

where we have defined $\Xi_{ij}(Y) = \frac{\partial \Xi_{ij}}{\partial Y_{ij}} Y$ and used the fact that $\Xi_{ij}(Y)$ is an antisymmetric matrix, $\Xi_{ij}(Y) + \Xi_{ij}(Y) = 0$, which in turn follows from the fact that $Y$ is an orthonormal matrix,

$$Y^T Y = I_{d_0} \Rightarrow \frac{\partial Y^T}{\partial Y_{ij}} Y + Y^T \frac{\partial Y}{\partial Y_{ij}} = \Xi_{ij}(Y) + \Xi_{ij}(Y) = 0. \quad (11)$$

Taking the antisymmetric property of $\Xi_{ij}(Y)$ into account in (10), we obtain

$$\frac{\partial}{\partial Y_{ij}} \log \lambda_k = Y_{ik} Y_{jk}, \quad (12)$$

$$\Xi_{ij}(Y_{kl}) = \frac{Y_{ik} Y_{jl} + Y_{il} Y_{jk}}{2(\log \lambda_l - \log \lambda_k)}, \quad \forall l \neq k. \quad (13)$$

It is notable that by construction, we do not have repeating eigenvalues, that is $\lambda_k \neq \lambda_l$, $\forall k \neq l$, so there exists a unique solution to (13). Once $\Xi_{ij}(Y)$ is computed, it follows that

$$\frac{\partial Y}{\partial Y_{ij}} = Y \Xi_{ij}(Y), \quad \frac{\partial Y^T}{\partial Y_{ij}} = -\Xi_{ij}(Y) Y^T. \quad (14)$$

In summary, the necessary partial derivatives for computing (9) is given by (12) and (14). Once (9) is computed for all $i, j$, we can evaluate (7) and ultimately evaluate (6).

**Optimization.** The remaining steps are feed-forward computation, backpropagation, and learning, as in the standard MLP. However, here, the backpropagation requires taking derivatives with respect to the matrix functions. These steps are described in Appendix C.

### 3.2 The General Form of the mMLP

We now discuss a general version of the mMLP which produces both a vector and an SPD matrix as outputs. An important application of this model is in heteroscedastic multivariate regression which we will discuss in Section 5.2.

**Model construction.** As before, let $X \in \mathbb{R}^{p_1 \times p_2}$ denote the input matrix. The corresponding outputs in this case are: an SPD matrix of trace one $Y \in \mathbb{R}^{d_0 \times d_0}$ and $y \in \mathbb{R}^{r_0}$. The mMLP of $L$ hidden layers is denoted by mMLP: $X \to \{\hat{y}, \hat{Y}\}$ and constructed as:

$$\begin{aligned}
\hat{y} &= h(z_0), & z_0 &= C_0 \hat{Y} \Lambda_0 h_1 + b_0, \\
\hat{Y} &= \mathcal{H}(Z_0), & Z_0 &= W_0 H_1 W_0^T + B_0, \\
h_l &= h(z_l), & z_l &= C_l H_l A_l h_{l+1} + b_l, \quad \forall 1 \leq l \leq L, \\
H_l &= \mathcal{H}(Z_l), & Z_l &= W_l H_{l+1} W_l^T + B_l, \quad \forall 1 \leq l \leq L, \\
h_{L+1} &= h(z_{L+1}), & z_{L+1} &= C_{L+1} H_{L+1} A_{L+1} h_{L+1} + b_{L+1}, \\
H_{L+1} &= \mathcal{H}(Z_{L+1}), & Z_{L+1} &= W_{L+1} \text{vec}(X(W_{L+1} h_{L+1} h_{L+1}^T + B_{L+1})^T) + B_{L+1},
\end{aligned} \quad (15)$$

where $h_l \in \mathbb{R}^{r_l}$, $H_l \in \mathbb{R}^{d_l \times d_l}$, $\forall 1 \leq l \leq L$, $z_l, b_l \in \mathbb{R}^{r_l}$, $Z_l, B_l \in \mathbb{R}^{d_l \times d_l}$, $C_l \in \mathbb{R}^{r_l \times r_l}$, $\forall 0 \leq l \leq L+1$, $A_l \in \mathbb{R}^{d_l \times r_{l+1}}$, and $W_l \in \mathbb{R}^{d_l \times d_{l+1}}$, $\forall 0 \leq l \leq L$. Just as in the standard MLP, $h$ is an activation function of choice, e.g., the hyperbolic tangent function.
**Loss function.** The loss function here needs to be designed with the specific application in mind. In the case of the heteroscedastic multivariate regression, the loss can be defined based on the log-likelihood. This will be discussed in Section 4.

**Optimization.** The remaining steps of feed-forward computation, backpropagation, and learning, are all described in Appendix D.

### 4 THE mMLP IN HETEROSEDASTIC MULTIVARIATE REGRESSION

In this section, we discuss an application of the mMLP in relation to the heteroscedastic multivariate regression, more specifically, the joint mean-covariance regression task.

**Model construction.** Let $D_{\text{train}} = \{y^{(i)}, x^{(i)}\}_{i=1}^n$ be our training dataset consisting of a set of inputs $x_i \in \mathbb{R}^{d_x}$ and a set of responses $y_i \in \mathbb{R}^{d_y}$. Consider the following multivariate regression problem: $y^{(i)} = f(x^{(i)}) + e^{(i)}$, where $f$ is a nonlinear function, and $e^{(i)}$ is the additive noise on the $i$-th response measurement. The goal is estimation of the conditional mean function $E[y \mid x]$ and the conditional covariance function $\text{Var}[y \mid x]$ for an unseen input $x_* \in D_{\text{test}}$.

We consider two likelihood models based on our choices of the noise model, namely, the multivariate Gaussian model and its generalization the multivariate power exponential model.

**Multivariate Gaussian model.** We first consider the noise model to follow a zero-mean multivariate Gaussian (mG) distribution with a dense covariance matrix, that is $e_i \sim \mathcal{N}(0, \Sigma_i)$. Let $\Sigma_i = \eta_i \Omega_i$, where $\eta_i = \text{tr}(\Sigma_i)$ and $\Omega_i$ is a trace-one matrix. The noise model can accordingly be reformulated as $e_i \sim \mathcal{N}_{\text{tr}1}(0, \Omega_i, \eta_i)$ where $\mathcal{N}_{\text{tr}1}$ is referred to as the trace-one mG distribution. Although these two formulations of the mG distribution are identical, we find it easier to work with the latter. This is because the output layer of the mMLP model in (15) operates under the trace-one constraint. It is of course possible to drop the trace-one constraint from the output layer, but we would then also be forced to use a different kernel function than the Mercer sigmoid kernel in that layer. Instead, we find it easier to work with this reformulated mG distribution with a trace-one covariance matrix which allows us to use the same choice of kernel function across all layers.

Given $e_i \sim \mathcal{N}(0, \Omega_i, \eta_i)$, the likelihood is defined as

$$\ell_{\mathcal{N}_{\text{tr}1}} := \log \mathcal{N}_{\text{tr}1}(y; \mu, \Omega, \eta), \quad \text{mMLP}_\theta : x \rightarrow \{\mu, \Omega, \eta\},$$

where $E[y \mid x] = \mu$, and $\text{Var}[y \mid x] = \eta \Omega$. The set $\theta$ includes all the neural network parameters.

**Multivariate power exponential model.** We next consider the noise model to follow a zero-mean multivariate power exponential (mPE) distribution (e.g., Gómez et al., 1998) which is a generalized variant of the mG distribution, that is $e_i \sim \mathcal{E}(0, \Sigma_i, \alpha_i, \beta_i)$ where $\Sigma_i$ is a symmetric real dispersion matrix and the pair of $\alpha_i \in \mathbb{R}^+$ and $\beta_i \in \mathbb{R}^+$ control the tail and the shape of the distribution. As a special case, for $\alpha = 1$ and $\beta = 1$, the mPE includes the mG distribution.

As in the case of the mG distribution, we find it easier to work with a reformulated variant where the dispersion matrix is of trace-one. Let $\Sigma_i = \eta_i \Omega_i$, where $\eta_i = \text{tr}(\Sigma_i)$. The noise model can accordingly be represented as $e_i \sim \mathcal{E}_{\text{tr}1}(0, \Omega_i, \alpha_i, \beta_i, \eta_i)$ where $\mathcal{E}_{\text{tr}1}$ is referred to as the trace-one mPE distribution.

Given $e_i \sim \mathcal{E}_{\text{tr}1}(0, \Omega_i, \alpha_i, \beta_i, \eta_i)$, the likelihood is defined as:

$$\ell_{\mathcal{E}_{\text{tr}1}} := \log \mathcal{E}_{\text{tr}1}(y; \mu, \Omega, \alpha, \beta, \eta), \quad \text{mMLP}_\theta : x \rightarrow \{\mu, \Omega, \alpha, \beta, \eta\},$$

where $E[y \mid x] = \mu$ and $\text{Var}[y \mid x] = \eta \alpha \nu(\beta) \Omega$, where $\nu(\beta) = \frac{2^{1/\beta} \Gamma(\frac{d_y}{2\beta})}{\Gamma(\frac{d_y}{2})}$. The set $\theta$ includes all the neural network parameters.

**Optimization.** Finally, the optimization involves learning the neural network parameters $\theta$ by maximizing the likelihoods given by (16) and (17).

---

1Refer to Appendix F.1 for the exact functional form of the trace-one mG distribution.
2Figure G.1 visualizes the probability density function of the distribution for selected values of $\alpha$ and $\beta$.
3Refer to Appendix G for the exact functional form of the trace-one mPE distribution and its basic properties.
5 EXPERIMENTS

Experiments are divided into two parts. The first part is on the empirical validation of the mMLP model in a supervised task of learning SPD matrices using synthetic data. The second part discusses our results in heteroscedastic multivariate regression on real-world datasets.

5.1 SPD MATRIX LEARNING

5.1.1 THE CHOICE OF LOSS FUNCTION

Consider the problem of learning SPD matrices on synthetic data using the mMLP model of (3). The objectives are to validate the model, and to evaluate the effect of the choice of the loss function on the overall performance.

The following loss functions are considered for this analysis: The first candidate is the loss function based on the normalized von Neumann divergence $\ell_{QRE}(\hat{Y}, Y)$ given by (5). The $\ell_{QRE}$ is related to the Riemannian geometry. The second candidate is the quadratic loss, $\ell_{quad}(\hat{Y}, Y) = \text{tr}((\hat{Y} - Y)(\hat{Y} - Y)^\top)$, which is related to the Euclidean geometry.

**Example 1.** Consider the set $D_{\text{train}} = \{x_i, Y_i\}_{i=1}^{n_{\text{train}}}$ of inputs $x_i \in \mathbb{R}^{20}$ and corresponding SPD matrix outputs $Y_i \in \mathbb{R}^{d_0 \times d_0}$ which are in this case dense covariance matrices (refer to Appendix H.1 for details on the data generation). The goal is to estimate the covariance matrices $\hat{Y}$ associated to the input vectors from the unseen test set $D_{\text{test}} = \{x_i\}_{i=1}^{n_{\text{test}}}$. The training size is varied between $n_{\text{train}} = \{10^2, 10^4\}$ samples. The analysis is carried out for $d_0 = \{10, 20\}$. Two examples of the test target outputs $Y_i$ for $d_0 = 20$ and $n_{\text{train}} = 10^2$ are visualized in Figure 1-A.

The mMLP models (3) are trained using 3 layers (20 units per layer, $d_l = 20$) under our two choices of loss functions: $\ell_{QRE}$, and $\ell_{quad}$. All models share the same initialization and the only difference here is the loss function. Refer to Appendix H.1 for additional details on the mMLP initialization.

The performance is evaluated on the test set, $n_{\text{test}} = 10^3$, in terms of the losses as the error measures, shown as $E_{QRE}$ and $E_{quad}$. Table 1 summarizes the results of the evaluation (also refer to Figure 1-B,C for the visual illustration). The key observation is that the quality of estimates differs considerably depending on the choice of the loss function. The loss function $\ell_{QRE}$ that takes into account the geometry of the SPD matrices clearly outperforms the one based on the Euclidean geometry, $\ell_{quad}$. The advantage is more pronounced for small set of training data.

5.1.2 THE mMLP VS. THE MLP BASED ON CHOLESKY DECOMPOSITION

As we discussed in Section 1, one can take a heuristic approach and tailor the output layer of a vanilla MLP so that the resulting output matrix satisfies the SPD requirement.

For the sake of comparison, we solve the same task as in Example 1 using the standard MLP with 3 layers (200 units per layer) and with the quadratic loss. To meet the SPD requirement, we use the
Table 1: SPD matrix learning on synthetic dataset (Example 1).

<table>
<thead>
<tr>
<th>Model/Loss</th>
<th>$d_0 = 10, n_{\text{train}} = 10^4$</th>
<th>$d_0 = 20, n_{\text{train}} = 10^4$</th>
<th>$d_0 = 10, n_{\text{train}} = 10^2$</th>
<th>$d_0 = 20, n_{\text{train}} = 10^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>mMLP/E quad</td>
<td>$E_{\text{quad}}$</td>
<td>$E_{\text{QRE}}$</td>
<td>$E_{\text{quad}}$</td>
<td>$E_{\text{QRE}}$</td>
</tr>
<tr>
<td>mMLP/E quad</td>
<td>$2.1 \times 10^{-3}$</td>
<td>$2.1 \times 10^{-5}$</td>
<td>$3.7 \times 10^{-2}$</td>
<td>$6.0 \times 10^{-2}$</td>
</tr>
<tr>
<td>MLP/E quad</td>
<td>$2.5 \times 10^{-3}$</td>
<td>$3.4 \times 10^{-1}$</td>
<td>$6.5 \times 10^{-1}$</td>
<td>$7.3 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

Table 2: Shallow vs deep architecture, Example 2. Error is measured in terms of $E_{\text{QRE}}$.

<table>
<thead>
<tr>
<th>Model</th>
<th>$d_0 = 10$</th>
<th>$d_0 = 20$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep architecture</td>
<td>$L = 2$</td>
<td>$L = 4$</td>
</tr>
<tr>
<td>Shallow architecture</td>
<td>$4 \times 10^{-3}$</td>
<td>$5 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

5.1.3 Shallow vs Deep SPD Matrix Learning

The design of the mMLP model in (3) enables a mechanism for deep SPD matrix learning by satisfying the SPD constraint across all input, hidden and output layers. The simpler approach would be to consider the standard MLP architecture across input and hidden layers but make use of the activation matrix functions only at the output layer to meet the SPD requirement:

\begin{equation}
\begin{aligned}
\hat{Y} &= \mathcal{H}(Z_0), & Z_0 &= W_0 h_1(W_0 1)^T + B_0, \\
\mathbf{h}_l &= h(z_l), & z_l &= W_l h_{l+1} + b_l, & 1 \leq l \leq L, \\
\mathbf{h}_{L+1} &= h(z_{L+1}), & z_{L+1} &= W_{L+1} \mathbf{vec}X + b_{L+1}.
\end{aligned}
\end{equation}

This amounts to a shallow design in the sense that it does not enable a mechanism for preserving the SPD constraint across all layers during the learning.

The design in (3) allows nonlinearities to pass through layers via activation function matrices imposing the SPD constraint, whereas in the shallow design, nonlinearities are propagated across layers via activation functions without imposing any constraints. Our hypothesis is that the former has advantage over the latter in that it captures complex dependencies which are important for the SPD matrix learning. Below, we present a numerical example which indeed highlights the importance of preserving the SPD constraint across all layers when learning the SPD matrix.

Example 2. Consider a similar experiment as in Example 1 for the case of $n_{\text{train}} = 10^2$ and output dimensions $d_0 = \{10, 20\}$. We directly compare the performance of (3) against (18) under different number of hidden layers $L = \{2, 4, 6\}$. For the mMLP model, the number of hidden units at each layer is set to 20, and for the MLP model it is set to 200 units. The shallow design (18) uses the hyperbolic tangent as the activation function $h(\cdot)$. The same choice of the activation function $\mathcal{H}(\cdot)$, given by (2), is used for both models. We use $E_{\text{QRE}}$ as the choice of the loss function for both models (refer to Appendix H.2 for additional details on the initialization). The performance is evaluated in terms of $E_{\text{QRE}}$.

Table 2 summarizes the results of the evaluation. Although the shallow design (18) performs relatively well, it underperforms in comparison to (3). Given the limited number of training samples, arbitrarily increasing the number of layers may not be necessarily advantageous, which is the case for both models. However, in this regard, the design in (18) is more sensitive.

5.2 Experimental Results on the Multi-output Regression Datasets

In heteroscedastic multivariate regression, the central task is to estimate the conditional covariance matrix that captures the spatial variations in correlations between the elements of the response vector. The underlying hypothesis is that if a model can effectively capture the heteroscedasticity, then it will provide better uncertainty quantification, and the estimation of the conditional mean response should also improve, in comparison to the model that is build based on the homoscedasticity assumption.
We showed the effectiveness of the proposed model on multiple real-world datasets. We introduced a new method to learn SPD matrices, referred to as the matrix multilayer perceptron (mMLP). We believe that there are many cases in which the mMLP can prove to be useful. An interesting direction for future work is to investigate application of the model in the context of heteroscedastic multivariate regression. Although it facilitates the implementation, we would need access to the full Jacobian matrices. Future work is needed in this direction.

Our implementation of the matrix backpropagation involves the use of multiple Kronecker products. Although it facilitates the implementation, we would need access to the full Jacobian matrices. However, these matrices are in fact available in the form of sparse block matrices, which means that it is possible to implement a memory-efficient computation of the tensor products without the need to actually have access to the full matrices. Future work is needed in this direction.

We believe that there are many cases in which the mMLP can prove to be useful. An interesting direction for future work is to investigate application of the model in the context of heteroscedastic multivariate regression. We showed the effectiveness of the proposed model on multiple real-world datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Appendix</th>
<th>$d_x$</th>
<th>$d_y$</th>
<th>$n_{\text{train}}$</th>
<th>$n_{\text{test}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocs10</td>
<td>\text{\cite{ocsmatlib}}</td>
<td>218</td>
<td>16</td>
<td>50</td>
<td>352</td>
</tr>
<tr>
<td>edm</td>
<td>\text{\cite{edm}}</td>
<td>2</td>
<td>16</td>
<td>50</td>
<td>103</td>
</tr>
<tr>
<td>ap7d</td>
<td>\text{\cite{ap7d}}</td>
<td>411</td>
<td>6</td>
<td>33</td>
<td>303</td>
</tr>
<tr>
<td>ap1d</td>
<td>\text{\cite{ap1d}}</td>
<td>411</td>
<td>6</td>
<td>100</td>
<td>236</td>
</tr>
<tr>
<td>scm1d</td>
<td>\text{\cite{scm1d}}</td>
<td>280</td>
<td>16</td>
<td>100</td>
<td>8802</td>
</tr>
<tr>
<td>scm20d</td>
<td>\text{\cite{scm20d}}</td>
<td>64</td>
<td>16</td>
<td>100</td>
<td>8802</td>
</tr>
</tbody>
</table>

For this purpose, we compare our mMLP-based regression models against another heteroscedastic-based mean-covariance regression model by Fox & Dunson (2015) and two homoscedastic-based mean-regression models, namely, the MLP regressor and the Gaussian process (GP). The models used in this experiment are summarized in Table 3.

We compare the performance of the heteroscedastic and homoscedastic regression models on six real-world multi-output datasets. Key features of the datasets are summarized in Table 4. The result suggests that the mMLP homoscedastic-based regression models are capable of capturing dependencies between the output measurement, which contributes to the better estimation of the mean predictions.

### 6 LIMITATIONS AND FUTURE WORK

The main limitation of the mMLP has to do with scalability to higher dimensions. The complexity associated with computing the $\alpha$-derivative of the von Neumann loss function at the output layer is $O(d_1^2)$. Taking the symmetric nature of the SPD matrices into account, the computational complexity at the hidden layer $l$ reduces to $O(d_l^2)$.

Our implementation of the matrix backpropagation involves the use of multiple Kronecker products. Although it facilitates the implementation, we would need access to the full Jacobian matrices. However, these matrices are in fact available in the form of sparse block matrices, which means that it is possible to implement a memory-efficient computation of the tensor products without the need to actually have access to the full matrices. Future work is needed in this direction.

We believe that there are many cases in which the mMLP can prove to be useful. An interesting direction for future work is to investigate application of the model in the context of the conditional density estimation within the framework of variational autoencoders.

### 7 DISCUSSION

We introduced a new method to learn SPD matrices, referred to as the matrix multilayer perceptron (mMLP). The mMLP takes the non-Euclidean geometry of the underlying SPD manifolds into account by making use of the von Neumann divergence as the choice of the SPD manifold metric. One key aspect of the mMLP is that it preserves the SPD constraint across all layers by exploiting PD kernel functions and a backpropagation algorithm that respects the inherent SPD nature of the matrices. We studied application of the model in the context of heteroscedastic multivariate regression. We showed the effectiveness of the proposed model on multiple real-world datasets.

### Table 3: Summary of the regression models used in the experiments.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Source</th>
<th>Model specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>mMLP-mPE</td>
<td>\text{Section 4}</td>
<td>3 layers ($r_1 = 200$, $d_1 = 200$), Adam optimizer, activation functions as in Table 4.</td>
</tr>
<tr>
<td>mMLP-mG</td>
<td>\text{Section 4}</td>
<td>3 layers ($r_1 = 200$, $d_1 = 200$), Adam optimizer, activation functions as in Table 4.</td>
</tr>
<tr>
<td>MLP</td>
<td>\text{Goodfellow et al. (2016)}</td>
<td>3 layers (200 units), ReLU activation, Adam optimizer, and default initialization of the scikit-learn.</td>
</tr>
<tr>
<td>GP</td>
<td>\text{Rasmussen &amp; Williams (2006)}</td>
<td>RBF kernel and default initialization of the scikit-learn.</td>
</tr>
<tr>
<td>NBCR</td>
<td>\text{Fox &amp; Dunson (2015)}</td>
<td>$10^5$ Gibbs iterations (discarded the first half), truncation of 5, and default initialization of the scikit-learn.</td>
</tr>
</tbody>
</table>

### Table 4: Choice of activation functions for the mMLP-mPE and mMLP-mG models in Table 3.

<table>
<thead>
<tr>
<th>Model</th>
<th>Output layer</th>
<th>Hidden layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mMLP-mPE</td>
<td>linear()</td>
<td>linear()</td>
</tr>
<tr>
<td>mMLP-mG</td>
<td>linear()</td>
<td>0.5+Sigmoid()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Source</th>
<th>Model specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>mMLP-mPE</td>
<td>\text{Section 4}</td>
<td>3 layers ($r_1 = 200$, $d_1 = 200$), Adam optimizer, activation functions as in Table 4.</td>
</tr>
<tr>
<td>mMLP-mG</td>
<td>\text{Section 4}</td>
<td>3 layers ($r_1 = 200$, $d_1 = 200$), Adam optimizer, activation functions as in Table 4.</td>
</tr>
<tr>
<td>MLP</td>
<td>\text{Goodfellow et al. (2016)}</td>
<td>3 layers (200 units), ReLU activation, Adam optimizer, and default initialization of the scikit-learn.</td>
</tr>
<tr>
<td>GP</td>
<td>\text{Rasmussen &amp; Williams (2006)}</td>
<td>RBF kernel and default initialization of the scikit-learn.</td>
</tr>
<tr>
<td>NBCR</td>
<td>\text{Fox &amp; Dunson (2015)}</td>
<td>$10^5$ Gibbs iterations (discarded the first half), truncation of 5, and default initialization of the scikit-learn.</td>
</tr>
</tbody>
</table>

### Table 5: Datasets and regression results.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$d_x$</th>
<th>$d_y$</th>
<th>$n_{\text{train}}$</th>
<th>$n_{\text{test}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocs10</td>
<td>218</td>
<td>16</td>
<td>50</td>
<td>352</td>
</tr>
<tr>
<td>edm</td>
<td>2</td>
<td>16</td>
<td>50</td>
<td>103</td>
</tr>
<tr>
<td>ap7d</td>
<td>411</td>
<td>6</td>
<td>33</td>
<td>303</td>
</tr>
<tr>
<td>ap1d</td>
<td>411</td>
<td>6</td>
<td>100</td>
<td>236</td>
</tr>
<tr>
<td>scm1d</td>
<td>280</td>
<td>16</td>
<td>100</td>
<td>8802</td>
</tr>
<tr>
<td>scm20d</td>
<td>64</td>
<td>16</td>
<td>100</td>
<td>8802</td>
</tr>
</tbody>
</table>

For this purpose, we compare our mMLP-based regression models against another heteroscedastic-based mean-covariance regression model by Fox & Dunson (2015) and two homoscedastic-based mean-regression models, namely, the MLP regressor and the Gaussian process (GP). The models used in this experiment are summarized in Table 3.

**Real-world datasets.** We compare the performance of the heteroscedastic and homoscedastic regression models on six real-world multi-output datasets. Key features of the datasets are summarized in Table 4. The performance is evaluated in terms of the root-mean-square error (RMSE) on test sets, shown in Table 5. The result suggests that the mMLP homoscedastic-based regression models are capable of capturing dependencies between the output measurements which contributes to the better estimation of the mean predictions.
REFERENCES


A  MATRIX NOTATIONS

We use \( \top \) for the transpose operator, \( \text{tr}(\cdot) \) for the trace operator, and \( \det(\cdot) \) for the matrix determinant. The symmetric part of a square matrix \( B \) is denoted by \( \text{sym}(B) = \left( B + B^\top \right)/2 \). The Kronecker product is denoted by \( \otimes \), the Hadamard product by \( \circ \), and the dot product by \( \cdot \). We use the vec-operator for column-by-column stacking of a matrix \( A \), shown as \( \text{vec}(A) \). Let \( A \) be an \( m \times n \) matrix, the operator \( P_{(m,n)} \) will then rearrange \( \text{vec}(A) \) to its matrix form: \( A = P_{(m,n)}(\text{vec}(A)) \). For the \( m \times n \) dimensional matrix \( A \), the commutation matrix is shown as \( K_{(m,n)} \) which is the \( mn \times mn \) matrix that transforms \( \text{vec}(A) \) into \( \text{vec}(A)^\top \): \( K_{(m,n)}\text{vec}(A) = \text{vec}(A)^\top \). An \( m \times m \) identity matrix is shown as \( I_m \). If \( T(X) : \mathbb{R}^{d\times d} \rightarrow \mathbb{R} \) is a real-valued function on matrices, then \( \nabla_X T(X) \) denotes the gradient with respect to the matrix \( X \), \( \nabla_X T(X) = \left[ \frac{\partial T(X)}{\partial X} \right]_{i,j=1:d} \). The matrix logarithm and the matrix exponential \( \exp(A) \) \( \log(A) \) are written as \( \log(A) \) and \( \exp(A) \), respectively. The matrix exponential in the case of symmetric matrices can be expressed using the eigenvalue decomposition as \( \exp(A) = V(\exp(\Lambda))V^\top \), where \( V \) is an orthonormal matrix of eigenvectors and \( \Lambda \) is a diagonal matrix with the eigenvalues on the diagonal. The matrix logarithm is the inverse of the matrix exponential if it exists. If \( A \) is symmetric and strictly positive definite (PD) it is computed using \( \log(A) = \log(V\Lambda V^\top) \), where \( (\log(A))_{i,i} = \log(A_{i,i}) \).

B  THE \( \alpha \)-DERIVATIVE: DEFINITION AND PROPERTIES

Definition  Let \( F \) be an \( m \times n \) matrix function of an \( n \times q \) matrix of variables \( X \). The \( \alpha \)-derivative of \( F(X) \) is defined as (Magnus, 2010, Definition 2)

\[
D_X F := \frac{\partial \text{vec}(F(X))}{\partial \text{vec}(X)} \quad \tag{19}
\]

where \( D_X F \) is an \( m\times p \times n \times q \) matrix which contains all the partial derivatives such that each row contains the partial derivatives of one function with respect to all variables, and each column contains the partial derivatives of all functions with respect to one variable.

Product rule  Let \( F \) \( (m \times p) \) and \( G \) \( (p \times r) \) be functions of \( X \) \( (n \times q) \). Then the product rule for the \( \alpha \)-derivative is given by (Magnus, 2010)

\[
D_X (FG) = (G^\top \otimes I_m)D_X F + (I_r \otimes F)D_X G \quad \tag{20}
\]

Chain rule  Let \( F \) \( (m \times p) \) be differentiable at \( X \) \( (n \times q) \), and \( G \) \( (l \times r) \) be differentiable at \( Y = F(X) \), then the composite function \( H(X) = G(F(X)) \) is differentiable at \( X \), and

\[
D_X H = D_Y GD_X F \quad \tag{21}
\]

which expresses the chain rule for the \( \alpha \)-derivative (Magnus, 2010).

C  THE BASIC CASE OF THE MMLP

C.1  FEEDFORWARD STEP

At the feed-forward computation, we compute and store the latent outputs \( \hat{Y}, H_l \) for all \( l \in \{L+1, \ldots, l\} \) using the current setting of the parameters, which are \( W_l \) and \( B_l \) computed from the learning step, Appendix C.3.

C.2  BACKPROPAGATION STEP

We first summarize the necessary \( \alpha \)-derivatives for the backpropagation, and then write down the backpropagation procedure accordingly.

Derivatives required for backpropagation  The derivative of the activation matrix function depends on the specific choice of kernel function, and in general it is computed readily from the
Via repeated use of the product rule of Appendix E,

\[
K = \text{vec}_l \frac{\partial \mathcal{H}(Z_l)}{\partial \text{vec}(Z_l)}^\top, \quad l \in \{0, \ldots, L + 1\}. \tag{22}
\]

For our specific choice of activation function, the Mercer Sigmoid kernel \(2\), it is computed in Appendix [2].

Via repeated use of the product rule of \(\alpha\)-derivatives \(20\), we obtain

\[
D_{\mathbf{W}} Z_l = (\mathbf{W} \otimes I_{d_l})(\mathbf{H}_{l+1}^\top \otimes I_{d_l}) + (I_{d_l} \otimes (\mathbf{W} I_{l+1})) K_{(d_l, d_{l+1})}, \quad l \in \{0, \ldots, L\}, \tag{23}
\]

\[
D_{\mathbf{W}_{L+1}} Z_l = (\mathbf{W}_{L+1} 1_{p_1 p_2} \otimes I_{d_{L+1}})((\mathbf{vec} X)^\top \otimes I_{d_{L+1}})
+ (I_{d_{L+1}} \otimes (\mathbf{W}_{L+1} \mathbf{vec} X))(I_{d_{L+1}} \otimes 1_{p_1 p_2}) K_{(d_{L+1}, 1_{p_1 p_2})}, \tag{24}
\]

\[
D_{\mathbf{H}_{l+1}} Z_l = (\mathbf{W} \otimes I_{d_l})(I_{d_{l+1}} \otimes \mathbf{W}^l), \quad l \in \{0, \ldots, L\}, \tag{25}
\]

where \(K\) is the commutation matrix (refer to Appendix [A] for a summary of the matrix notation).

**Backpropagation** In the interest of simple expressions, let \(\hat{\mathbf{H}}_0 = \hat{\mathbf{Y}}\). Backpropagation to the hidden layer \(l\) is computed recursively using the derivatives computed at the previous layer according to

\[
D_{\mathbf{Z}_l} \ell = D_{\mathbf{H}_l} \ell D_{\mathbf{Z}_l} \mathbf{H}_l, \quad \forall l \in \{0, \ldots, L + 1\}, \tag{26}
\]

\[
D_{\mathbf{W}_l} \ell = D_{\mathbf{Z}_l} \ell D_{\mathbf{W}_l} \mathbf{Z}_l, \quad \forall l \in \{0, \ldots, L + 1\}, \tag{27}
\]

\[
D_{\mathbf{H}_{l+1}} \ell = D_{\mathbf{Z}_l} \ell D_{\mathbf{H}_{l+1}} \mathbf{Z}_l, \quad \forall l \in \{0, \ldots, L\}, \tag{28}
\]

\[
D_{\mathbf{B}_l} \ell = D_{\mathbf{Z}_l} \ell, \quad \forall l \in \{0, \ldots, L + 1\}. \tag{29}
\]

**C.3 Learning step**

Learning involves updating the weights \(\mathbf{W}_l\) and the biases \(\mathbf{B}_l\) using derivatives computed during the backpropagation step. These are updated using derivatives \(D_{\mathbf{W}_l} \ell\) and \(D_{\mathbf{B}_l} \ell\) for a given learning rate \(\eta\) as

\[
\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta P_{(d_l, d_{l+1})}(D_{\mathbf{W}_l} \ell), \quad \forall l \in \{0, \ldots, L + 1\}, \tag{30}
\]

\[
\mathbf{B}_l \leftarrow \mathbf{B}_l - \eta P_{(d_l, d_{l})}(D_{\mathbf{B}_l} \ell), \quad \forall l \in \{0, \ldots, L + 1\}, \tag{31}
\]

where \(P\) is the rearrangement operator introduced in Appendix [A].

**D The general form of the MMLP**

**D.1 Feedforward step**

The forward path involves computing and storing both \(\mathbf{h}_l, \hat{\mathbf{y}}\) and \(\mathbf{H}_l, \hat{\mathbf{Y}}\) using the current settings of the parameters, for all \(l \in \{0, \ldots, L + 1\}\).

**D.2 Backpropagation step**

Most of the necessary derivatives are identical to the ones in Appendix [C.2]. However, there are some additional derivatives needed which we will discuss in the following. We then write down the backpropagation formula.

**D.2.1 Required derivatives for backpropagation**

The derivative of the activation function depends on the choice of the function, and it is computed using the definition of the \(\alpha\)-derivative,

\[
D_{\mathbf{z}_l} \mathbf{y}_l = \frac{\partial \mathbf{y}_l(\mathbf{z}_l)}{\partial \mathbf{z}_l}^\top, \quad l \in \{0, \ldots, L + 1\}. \tag{32}
\]
The learning step involves updating the weights and the biases which are computed using derivatives computed from the backpropagation step. Update rules for $W_l$ and $B_l$ are the same as the ones given in Appendix D.3. The remaining parameters are learned in a similar fashion,

$$A_l \leftarrow A_l - \eta P_{(d_i, r_{l+1})}(D A_l, \ell), \quad \forall l \in \{0, \ldots, L + 1\},$$

$$C_l \leftarrow C_l - \eta P_{(r_{l}, d_i)}(D C_l, \ell), \quad \forall l \in \{0, \ldots, L + 1\},$$

$$b_l \leftarrow b_l - \eta P_{(r_{l}, 1)}(D b_l, \ell), \quad \forall l \in \{0, \ldots, L + 1\}.$$

### E THE $\alpha$-DERIVATIVE OF THE MERCER SIGMOID KERNEL

The $\alpha$-derivative of the Mercer sigmoid kernel can be computed as

$$DZH = \begin{pmatrix} \cdots & \cdots & \cdots \\ \frac{\partial}{\partial z_i} \kappa_{mn} \tr K & \cdots \\ \cdots \end{pmatrix}, \quad \forall i, m, n \in \{1, \ldots, d_l\},$$

where $z_i$ indicates the $i^{th}$ column of $Z$, $\tr K \equiv \tr K(Z)$, $\kappa_{mn} \equiv \kappa(z_m, z_n)$ as defined in [8], and

$$\frac{\partial \kappa_{mn}}{\partial z_i \tr K} = \left(\frac{\alpha (1^T - \hat{f}(z_i)) \circ \hat{f}(z_i))}{\tr K^2}\right) \circ \left(\tr K \hat{f} \left(\frac{\partial}{\partial z_i} (z_m \circ z_n)\right) - 2 \kappa_{mn} \hat{f}(z_i)\right),$$

where $\hat{f}(z_i) := \tanh(\alpha z_i - \beta)$. 

The probability density function of a trace-one mPE distribution, $E_{tr1}(\mu, M/\eta, \alpha, \beta)$ for fixed values of $\mu, M, \eta = \text{tr}(M)$ and varying values of scale and shape parameters $(\alpha, \beta)$. When $\alpha = 1$ and $\beta = 1$, the density corresponds to the multivariate Gaussian distribution $N(\mu, M)$.

### F Trace-one multivariate Gaussian distribution

#### F.1 Probability density function

For a $d$-dimensional random variable $\boldsymbol{\vartheta} \in \mathbb{R}^d$, we define the trace-one Gaussian distribution according to

$$
N_{tr1}(\boldsymbol{\vartheta}; \mu, \Omega, \eta) = \frac{1}{\sqrt{\text{det}(2\pi\eta\Omega)}} e^{-\frac{1}{2}(\boldsymbol{\vartheta} - \mu)^\top (\eta\Omega)^{-1}(\boldsymbol{\vartheta} - \mu)},
$$

where $\mu \in \mathbb{R}^d$ is the mean, $\eta \in \mathbb{R}^+$ is the scale parameter, and $\Omega \in \mathbb{R}^{d \times d}$ is the trace-one covariance matrix, $\text{tr}(\Omega) = 1$.

#### F.2 The $\alpha$-derivatives of the log-pdf

The $\alpha$-derivatives of the trace-one Gaussian distribution’s log-pdf with respect to its parameters are summarized as

$$
D_\eta \ln N_{tr1}(\boldsymbol{\vartheta}; \mu, \Omega, \eta) = -\frac{1}{2}(\text{vec}(\Omega^{-1}))^\top - \frac{1}{2}(D_\Omega t)^\top,
$$

$$
D_\Omega t = -\text{vec}((\eta\Omega)^{-1}(\boldsymbol{\vartheta} - \mu)(\boldsymbol{\vartheta} - \mu)^\top \Omega^{-1}),
$$

$$
D_\mu \ln N_{tr1}(\boldsymbol{\vartheta}; \mu, \Omega, \eta) = (\boldsymbol{\vartheta} - \mu)(\eta\Omega)^{-1},
$$

$$
D_{\log\eta} \ln N_{tr1}(\boldsymbol{\vartheta}; \mu, \Omega, \eta) = -\frac{d}{2} + \frac{1}{2}(\boldsymbol{\vartheta} - \mu)^\top (\eta\Omega)^{-1}(\boldsymbol{\vartheta} - \mu).
$$

### G Trace-one multivariate power exponential (mPE) distribution

#### G.1 Probability density function

For a random variable $\boldsymbol{\vartheta} \in \mathbb{R}^d$, the trace-one mPE distribution can be expressed as

$$
E_{tr1}(\boldsymbol{\vartheta}; \mu, \Omega, \alpha, \beta) = \frac{c(\alpha, \beta)}{(\det(\eta\Omega))^\frac{d}{2}} \exp\left\{\frac{1}{2} t(\boldsymbol{\vartheta}; \mu, \Omega)^\top (\eta\Omega)^{-1} t(\boldsymbol{\vartheta}; \mu, \Omega)\right\},
$$

$$
c(\alpha, \beta) = \frac{\beta \Gamma\left(\frac{d}{2}\right)}{\pi^{\frac{d}{2}} \Gamma\left(\frac{d-2}{2}\right) \alpha^d}, \quad t(\boldsymbol{\vartheta}; \mu, \Omega) := (\boldsymbol{\vartheta} - \mu)^\top \Omega^{-1}(\boldsymbol{\vartheta} - \mu), \quad \text{tr}(\Omega) = 1.
$$

Here, $\mu$ is the mean vector, and $\Omega$ is a $d \times d$ symmetric real dispersion matrix where $\text{tr}(\Omega) = 1$. The parameter $\eta$ has the same role as in the trace-one Gaussian distribution. The pair of $\alpha \in \mathbb{R}^+$
and \( \beta \in \mathbb{R}^+ \) control the tail and the shape of the distribution. As a special case, the mPE includes the Gaussian distribution: For \( \alpha = 1 \) and \( \beta = 1 \), the trace-one mPE distribution corresponds to the trace-one multivariate Gaussian distribution. Figure 11 visualizes the probability density function of the distribution for selected values of \( \alpha \) and \( \beta \), for the case of \( d = 2 \).

**Remark:** Very large and very small values of \( \alpha \) and \( \beta \) might be undesirable as, for one, they pose numerical challenges. In practice, these parameters can be restricted within a range by choosing suitable output activation functions. In all experiments in this paper, we choose to bound them conservatively as: \( 0.5 \leq \alpha, \beta \leq 1.5 \), using the sigmoid function.

### G.2 Moments

Let \( \vartheta \in \mathbb{R}^d \) and \( \vartheta \sim \mathcal{E}_{\text{tr}1} (\mu, \Omega, \alpha, \beta, \eta) \). The mPE’s mean vector and covariance matrix are computed from:

\[
E[\vartheta] = \mu, \quad \forall \vartheta[\vartheta] = \alpha \eta \nu(\beta) \Omega, \quad \nu(\beta) := \frac{2^{1/\beta} \Gamma(d + 2/\beta)}{d \Gamma(d/\beta)},
\]

where \( \Gamma(\cdot) \) denotes the gamma function.

### G.3 The \( \alpha \)-Derivatives of the Log-Pdf

It is straightforward to take derivatives of the mPE’s log-pdf using the favorable generalization properties of the \( \alpha \)-derivative’s chain and product rules. These are summarized as:

\[
\begin{align*}
D_\Omega \log \mathcal{E}_{\text{tr}1} (\vartheta; \mu, \Omega, \alpha, \beta, \eta) &= -\frac{1}{2} (\text{vec}(\Omega^{-1}))^\top \frac{\beta}{2 \alpha \eta} (t/\alpha \eta)^{\beta - 1} D_\Omega t, \\
D_\Omega t &= -(\text{vec}(\Omega^{-1})(\vartheta - \mu)(\vartheta - \mu)^\top \Omega^{-1})^\top, \\
D_\mu \log \mathcal{E}_{\text{tr}1} (\vartheta; \mu, \Omega, \alpha, \beta, \eta) &= -\frac{\beta}{2 \alpha \eta} (t/\alpha \eta)^{\beta - 1} D_\mu t, \\
D_\mu t &= -2(\text{vec}((\vartheta - \mu)^\top \Omega^{-1})^\top)^\top, \\
D_{\log \eta} \log \mathcal{E}_{\text{tr}1} (\vartheta; \mu, \Omega, \alpha, \beta, \eta) &= -\frac{d}{2} + \frac{\beta t}{2 \alpha \eta} (t/\alpha \eta)^{\beta - 1}, \\
D_\alpha \log \mathcal{E}_{\text{tr}1} (\vartheta; \mu, \Omega, \alpha, \beta, \eta) &= -\frac{d}{2 \alpha} + \frac{\beta t}{2 \eta \alpha^2} (t/\alpha \eta)^{\beta - 1}, \\
D_\beta \log \mathcal{E}_{\text{tr}1} (\vartheta; \mu, \Omega, \alpha, \beta, \eta) &= D_\beta \log \mathcal{C}(\alpha, \beta) - \frac{1}{2} (t/\alpha \eta)^{\beta - 1} \log (t/\alpha \eta), \\
D_\beta \log \mathcal{C}(\alpha, \beta) &= \frac{1}{\beta} + \frac{d}{2 \beta^2} (\psi(d/2 \beta) + \log 2).
\end{align*}
\]

### H Additional Details on the Experiments

#### H.1 Example 1

**Data generation** Let \( A \in \mathbb{R}^{d_o \times d_o} \) be a matrix where each of its elements is generated from a standard normal distribution. The matrix \( A \) is kept fixed. The \( i \)-th class covariance \( Y_i \) is computed according to the following procedure:

1. Draw \( 10^4 \) samples from a known Gaussian distribution \( \mathcal{N}(\mu_i, \Sigma_i) \) with a unique mean \( \mu_i \in \mathbb{R}^{d_o} \) and a unique dense covariance matrix \( \Sigma_i \in \mathbb{R}^{d_o \times d_o} \).
2. Let \( t_j \) be a random sample from this Gaussian. For this sample, compute \( y_j = At_j \). For all \( 10^4 \) samples, collect the results into \( \mathbf{y} = \{y_j\}_{j=1}^{10^4} \).
3. Compute the sample covariance of \( \mathbf{y} \) and normalize the resulting covariance matrix to trace one, that is \( \mathbf{Y} \leftarrow \text{cov}(\mathbf{y})/\text{tr(cov(\mathbf{y}))} \).
**Initialization** All models use the same batch size (equal to 5), the same choice of activation matrix function, which is given by the Mercer sigmoid kernel \( \mathcal{K} \), and the same optimizer (the Adam optimizer \cite{Adam} with default settings).

**H.2 Example 2**

**Data generation** See the data generation procedure in Example 1.

**Initialization** Both models \( \text{[18]} \) and \( \text{[3]} \) use the same batch size (equal to 5), the same choice of loss function \( \text{[5]} \), and the same optimizer (the Adam optimizer \cite{Adam} with default settings). Both models use the same choice of the output activation matrix function, given by the Mercer sigmoid kernel \( \mathcal{K} \). The model in \( \text{[18]} \) uses the hyperbolic tangent as the activation function across the hidden layers, while \( \text{[3]} \) makes use of the same choice of the activation matrix function as in its output layer.

**H.3 Multi-output regression datasets**

**H.3.1 oes10**

The dataset oes10 was obtained from \cite{Spyromitros-Xioufis2016}. The Occupational Employment Survey (OES) datasets contain records from the years of 2010 (OES10) of the annual Occupational Employment Survey compiled by the US Bureau of Labor Statistics. As described in \cite{Spyromitros-Xioufis2016}, "each row provides the estimated number of full-time equivalent employees across many employment types for a specific metropolitan area". We selected the same 16 target variables as listed in \cite{Spyromitros-Xioufis2016}, Table 5. The remaining 298 variables serve as the inputs. Data samples were randomly divided into training and test sets (refer to Table 5a).

**H.3.2 edm**

The dataset edm was obtained from \cite{Karalic1997}. The electrical discharge machining (EDM) dataset contain domain measurements in which the workpiece surface is machined by electrical discharges occurring in the gap between two electrodes: the tool and the workpiece. Given the two input variables, gap control and flow control, the aim here is to predict the other 16 target variables representing mean values and deviations of the observed quantities of the considered machining parameters. Data samples were randomly divided into training and test sets (refer to Table 5a).

**H.3.3 atp1d AND atp7d**

The datasets atp1d and atp7d were obtained from \cite{Spyromitros-Xioufis2016}. The Airline Ticket Price (ATP) dataset includes the prediction of airline ticket prices. As described in \cite{Spyromitros-Xioufis2016}, the target variables are either the next day price, atp1d, or minimum price observed over the next seven days atp7d for 6 target flight preferences listed in \cite{Spyromitros-Xioufis2016}, Table 5. There are 411 input variables in each case. The inputs for each sample are values considered to be useful for prediction of the airline ticket prices for a specific departure date, for example, the number of days between the observation date and the departure date, or the boolean variables for day-of-the-week of the observation date. Data samples were randomly divided into training and test sets (refer to Table 5a).

**H.3.4 scm1d AND scm20d**

The datasets scm1d and scm20d were obtained from \cite{Spyromitros-Xioufis2016}. The Supply Chain Management (SCM) datasets are derived from the Trading Agent Competition in Supply Chain Management (TAC SCM) tournament from 2010. As described in \cite{Spyromitros-Xioufis2016}, each row corresponds to an observation day in the tournament. There are 280 input variables in these datasets which are observed prices for a specific tournament day. The datasets contain 16 regression targets, where each target corresponds to the next day mean price scm1d or mean price for 20 days in the future scm20d for each product \cite{Spyromitros-Xioufis2016}, Table 5. Data samples were randomly divided into training and test sets (refer to Table 5a).