SIAM/ASA J. Uncertainty Quantification
Vol. 11, No. 1, pp. 62–90

On the Deep Active-Subspace Method*

Wouter Edeling†

Abstract. The deep active-subspace method is a neural-network based tool for the propagation of uncertainty through computational models with high-dimensional input spaces. Unlike the original active-subspace method, it does not require access to the gradient of the model. It relies on an orthogonal projection matrix constructed with Gram–Schmidt orthogonalization to reduce the input dimensionality. This matrix is incorporated into a neural network as the weight matrix of the first hidden layer (acting as an orthogonal encoder), and optimized using back propagation to identify the active subspace of the input. We propose several theoretical extensions, starting with a new analytic relation for the derivatives of Gram–Schmidt vectors, which are required for back propagation. We also study the use of vector-valued model outputs, which is difficult in the case of the original active-subspace method. Additionally, we investigate an alternative neural network with an encoder without embedded orthonormality, which shows equally good performance compared to the deep active-subspace method. Two epidemiological models are considered as applications, where one requires supercomputer access to generate the training data.

Key words. deep active subspaces, high-dimensional uncertainty quantification, Gram–Schmidt derivative, sensitivity analysis, epidemiology, neural networks

MSC codes. 65J99, 46N30, 65Q30

DOI. 10.1137/21M1463240

1. Introduction.

Since computational models play an increasingly important role in society, it is important to realize that these models are subject to (substantial) uncertainty. One form of uncertainty is parametric in nature, as models contain input parameters which are usually only known to an approximate degree. The uncertainty in the input values will propagate to the predictions of the model, rendering them uncertain as well.

Before the outcome of the model is acted upon, the (parametric) uncertainty should therefore first be assessed. Hence, we will focus on so-called forward uncertainty-quantification problems, where a probability distribution on the input is assumed, and the goal is to assess the corresponding distribution of the model outputs. A well-known technique, which scales well to high-dimensional input spaces, is Monte Carlo (MC) sampling. However, as MC sampling suffers from a slow convergence rate, other techniques have been developed. Examples include stochastic collocation (SC) and polynomial chaos (PC) expansions [17], which can show exponential convergence, therefore requiring many fewer samples from (expensive)

* Received by the editors December 3, 2021; accepted for publication (in revised form) August 30, 2022; published electronically February 2, 2023.
https://doi.org/10.1137/21M1463240
Funding: This work was supported by European Union Horizon 2020 research and innovation programme grant agreement 800925 (VECMA project).

† Centrum Wiskunde & Informatica, Scientific Computing Group, Science Park 123, 1098XG Amsterdam, The Netherlands (wouter.edeling@cwi.nl).

computational models compared to MC sampling. However, this potential convergence rate is conditional on the regularity of the model outcome, and the number of considered input parameters. The curse of dimensionality is inherent in the tensor-product based construction of the SC and PC methods, limiting their application to (typically) no more than 5 parameters.

However, even when a model contains a large parameter set, it is likely that in practice not all parameters will be of equal importance. Often, a relatively low-dimensional effective input dimension exists, in which most of the output variation takes place. If one is able to identify this dimension, the cost of sampling can be brought down substantially. The aforementioned SC and PC type methods have their adaptive counterparts, which concentrate on more important input variables in an iterative manner, thereby creating an anisotropic sampling plan; see, e.g., [19 26 2]. These have been applied to computational models with $\mathcal{O}(10)$ parameters [16] or sometimes (simple) problems with $\mathcal{O}(100)$ inputs; see, e.g., [24]. Another class of methods are those based on the high-dimensional model representation framework [32]. Rather than trying to efficiently sample a high-dimensional input space, these methods break the problem up into a series of low-dimensional forward uncertainty-propagation problems. Adaptivity can be incorporated here as well [30].

All aforementioned adaptive methods are iterative in nature, and build the final sampling using several smaller (sequential) ensembles. Active-subspace methods [9] are different, and perform dimension reduction in the stochastic input space in a postprocessing step, using a single (MC) ensemble. The goal is to discover an important, linear, low-dimensional manifold embedded in the high-dimensional input space. Once the active subspace is identified, one can exploit it by creating surrogate models (cheap approximations of the original code), in the active (reduced) input dimension. Active subspaces have been found in a wide range of problems, for instance in hydrology [27], hypersonic aerodynamics [8], airfoil shape optimization [20], and epidemiology [29]. Interesting connections between active subspaces and dimensional analysis have been made [11], and they have also been used to compress the size of neural networks [10].

To find the active subspace, the gradient of the model output with respect to the input parameters is required. In some simple cases this can be done analytically. Other options include finite differences or adjoint solvers. However, an adjoint solver might not be available and sampling enough code evaluations to compute the finite-difference derivatives might become prohibitively expensive in high dimensions. Other options include gradient sketching [7], or the use of an active-subspace approach that does not require direct access to gradient data to begin with. The focus of our investigation falls in the latter category. So-called deep active subspaces are introduced in [37], which combine active-subspace ideas with neural networks. The high-dimensional input vector, which forms the input layer of the neural network, is projected to a low-dimensional subspace via an orthogonal projection matrix based on Gram–Schmidt orthogonalization of an unconstrained matrix. This projection matrix forms the weight matrix of the first hidden layer, which acts as an orthonormal encoder. Since Gram–Schmidt is fully differentiable, back propagation can still be applied to minimize the (squared) loss function of the neural network.

To our knowledge, the deep active-subspace method has to date only been applied to relatively simple uncertainty-quantification problems in [37]. Our main goal with this paper is twofold. First, we apply the method to more demanding problems (in epidemiology),

one of which requires access to a supercomputer. Second, we extend the method on various theoretical grounds. Instead of relying on automatic differentiation of Gram–Schmidt's constituent arithmetic operations, we derive a new analytic recurrence relation for the derivative of Gram–Schmidt vectors, which are required to train the neural network. We also investigate the need for orthonormality in the first hidden layer, by contrasting the performance of the deep active-subspace method to a similar neural network without embedded orthonormality. Another subject of study is vector-valued quantities of interest, which are difficult in the case of the original active-subspace method. Finally, we also show how we can easily extract derivative-based global sensitivity metrics from the neural networks.

This article is organized as follows. The original and deep active-subspace methods are described in sections 2 and 3. Section 4 discusses the alternative neural network without an orthonormal weight matrix. Vector-valued quantities of interest are covered next, and in section 6 we describe the sensitivity analysis method. This is followed by a brief description of an alternative derivative-free active-subspace approach in section 7. Finally, sections 8 and 9 contain our results and the conclusion.

2. Active subspaces.

Let $f(\mathbf{x})$ represent our quantity of interest (QoI), i.e., the output of some (expensive) computational model, where $\mathbf{x} = [x_1, x_2, \dots, x_D]^T$ is a D dimensional vector of continuous, imperfectly know input parameters. We will assume f is a scalar QoI, unless specified otherwise. The uncertainty in $\mathbf{x}$ is specified by a joint probability-density function $p(\mathbf{x})$, which we assume is given.

The active-subspace method, introduced by [9], is a method for forward propagation of uncertainty in high-dimensional input spaces. Since we generally cannot assume that $f(\mathbf{x})$ will show the greatest variation in a direction that is exactly aligned with the coordinate axes of $\mathbf{x}$, the active-subspace method attempts to find a rotated coordinate system that is aligned with the directions along which f varies the most on average. A low-dimensional approximation of f is then created by only retaining the $d < D$ directions of greatest variability. To find these directions, the following gradient matrix is constructed:

$$C = \mathbb{E}\left[\left(\nabla f\left(\mathbf{x}\right)\right)\left(\nabla f\left(\mathbf{x}\right)\right)^T\right] = \int \left(\nabla f\left(\mathbf{x}\right)\right)\left(\nabla f\left(\mathbf{x}\right)\right)^T p(\mathbf{x})\mathbf{dx}. \tag{2.1}$$

Since C is an (uncentered) covariance-like matrix, it is symmetric positive semidefinite and has the following spectral decomposition,

$$C = U\Lambda U^T = [U_1\ U_2]\begin{bmatrix}\Lambda_1 & 0\\ 0 & \Lambda_2\end{bmatrix}[U_1\ U_2]^T, \tag{2.2}$$

with real eigenvalues $\lambda_i \geq 0$ contained in the diagonal matrices $\Lambda_1 := \mathrm{diag}\left(\lambda_1, \dots, \lambda_\mathrm{d}\right)$ and $\Lambda_2 := \mathrm{diag}\left(\lambda_{d+1}, \dots, \lambda_D\right)$. The eigenvalues are ordered as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$ such that Λ_1 contains the d largest eigenvalues and the column vectors of U_1 point in the direction of largest (on-average) variability. As such, most of the variability of f is retained along directions obtained by linearly projecting the input $\mathbf{x} \in \mathbb{R}^D$ to a low-dimensional 'active' subspace $\mathbf{y} \in \mathbb{R}^d$ via the tall-and-skinny matrix $U_1 \in \mathbb{R}^{D\times d}$ of orthonormal basis vectors, such that $U_1^T U_1 = I_d$, where I_d is the d-dimensional identity matrix. The active subspace is thus given by

$$\mathbf{y} = U_1^T \mathbf{x}, \tag{2.3}$$

and in a similar vein $\mathbf{z} = U_2^T\mathbf{x}$ are the "inactive" variables along which f varies relatively little. Lemma 2.2 from [9] relates the gradients of f with respect to $\mathbf{y}$ and $\mathbf{z}$ to the eigenvalues of Λ_1 and Λ_2, respectively:

$$\begin{aligned} \mathbb{E}\left[(\nabla_{\mathbf{y}} f)^T \nabla_{\mathbf{y}} f\right] &= \operatorname{trace}(\Lambda_1), \\ \mathbb{E}\left[(\nabla_{\mathbf{z}} f)^T \nabla_{\mathbf{z}} f\right] &= \operatorname{trace}(\Lambda_2). \end{aligned} \tag{2.4}$$

If all eigenvalues of Λ_2 are zero, (2.4) implies that $\nabla_{\mathbf{z}} f$ is zero everywhere in the stochastic domain. Such a function is called "$\mathbf{z}$-invariant," and we will show that the deep active-subspace method described later is $\mathbf{z}$-invariant by construction. Active-subspace methods approximate $f(\mathbf{x}) = f\left(UU^T\mathbf{x}\right) = f\left(U_1U_1^T\mathbf{x} + U_2U_2^T\mathbf{x}\right) = f\left(U_1\mathbf{y} + U_2\mathbf{z}\right)$ via a conditional expectation,

$$f(\mathbf{x}) \approx G(\mathbf{y}) = \mathbb{E}_{\mathbf{z}}[f \mid \mathbf{y}] = \int f(U_1\mathbf{y} + U_2\mathbf{z})\, p(\mathbf{z} \mid \mathbf{y})\, \mathrm{d}\mathbf{z} \approx \frac{1}{N}\sum_{i=1}^{N} f(U_1\mathbf{y} + U_2\mathbf{z}_i). \tag{2.5}$$

While (2.5) may describe a high-dimensional integral, if f is (nearly) $\mathbf{z}$-invariant, its MC approximation shown on the right will only require a very small number of samples, e.g., $N = 1$ [9].

Finally, note that our goal is to create an efficient surrogate model for the code output $f(\mathbf{x})$ using active-subspace ideas. To differentiate between the output of the code and the surrogate, we introduce the notation $\widetilde{f}(\mathbf{x})$, $\widetilde{G}(\mathbf{y})$ for the latter.

2.1. Finding U_1. The approach described above is intuitive, and has nice theoretical properties, such as error bounds; see [9]. The downside, however, is that the gradient $\nabla f(\mathbf{x})$ must be available. This downside has prompted the development of other active-subspace methods which do not require access to the gradient. Some of these methods involve Gaussian processes [28 38] or linear regression [4], whereas others use deep learning. We will mainly focus on the latter.

3. Deep active subspaces. In [37], an approach is described in which artificial neural networks (ANNs) are used to construct $\widetilde{G}(\mathbf{y})$, and where the equivalent to U_1 (denoted by W_1), is found using stochastic gradient descent and back propagation. Like the classical active-subspace method, the column vectors of W_1 still form an orthonormal basis. The difference is that the column vectors of W_1 are no longer the eigenvectors of the gradient matrix C, but instead are constructed using Gram–Schmidt orthogonalization. As such, W_1 is parametrized by an unconstrained matrix Q of the same dimension ($Q \in \mathrm{R}^{D\times d}$), where the nonorthogonal (yet independent) column vectors $\mathbf{q}_i \in \mathbb{R}^D$ are made orthogonal via

$$\mathbf{w}_i = \mathbf{q}_i - \sum_{j=1}^{i-1}\left(\frac{\mathbf{w}_j^T\mathbf{q}_i}{\mathbf{w}_j^T\mathbf{w}_j}\right)\mathbf{w}_j, \quad i = 1,\ldots,d. \tag{3.1}$$

That is, we start with $\mathbf{w}_1 := \mathbf{q}_1$, and for all subsequent vectors $\mathbf{q}_i$ we subtract the projections of $\mathbf{q}_i$ onto each vector $\mathbf{w}_j$ which has previously been orthogonalized. This leaves us with an orthogonal basis $[\mathbf{w}_1(\mathbf{q}_1)\ \ \mathbf{w}_2(\mathbf{q}_1,\mathbf{q}_2)\ \ \cdots\ \ \mathbf{w}_d(\mathbf{q}_1,\mathbf{q}_2,\ldots,\mathbf{q}_d)]$. Finally, to obtain an

orthonormal basis, each column vector is divided by its length, such that our final weight matrix becomes

$$W_1(Q) = \left[\frac{\mathbf{w}_1(\mathbf{q}_1)}{\|\mathbf{w}_1(\mathbf{q}_1)\|_2} \quad \frac{\mathbf{w}_2(\mathbf{q}_1, \mathbf{q}_2)}{\|\mathbf{w}_2(\mathbf{q}_1, \mathbf{q}_2)\|_2} \quad \cdots \quad \frac{\mathbf{w}_d(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_d)}{\|\mathbf{w}_d(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_d)\|_2} \right]. \tag{3.2}$$

While we experienced no issues in our computations, it should be noted that Gram–Schmidt orthogonalization can be numerically unstable; see, e.g., [25] for more information and mitigation strategies.

Note that the projection $\mathbf{y} = \Phi\left(W_1^T \mathbf{x}\right) = W_1^T \mathbf{x}$ also occurs in a layer of a neural network if the activation function $\Phi(\cdot)$ is linear [1]. Thus, we can interpret W_1 as the weight matrix of the first hidden layer (with d neurons and linear activation), connected to an input layer through which $\mathbf{x}$ is passed. Each column vector $\mathbf{w}_i$ contains all the weights connecting the input layer to the ith neuron of the first hidden layer; see Figure 1. Since the first hidden layer has only d neurons, and its weight matrix is determined from a Gram–Schmidt procedure, we call this layer the deep active-subspace (DAS) layer.

The surrogate of $G(\mathbf{y})$ is the ANN from the DAS layer onward; see Figure 2. Each hidden layer has a weight matrix $W_i \in \mathbb{R}^{p+1 \times p}$, assuming that all hidden layers have p neurons plus 1 bias neuron. As per usual, these weight matrices are optimized through the back propagation

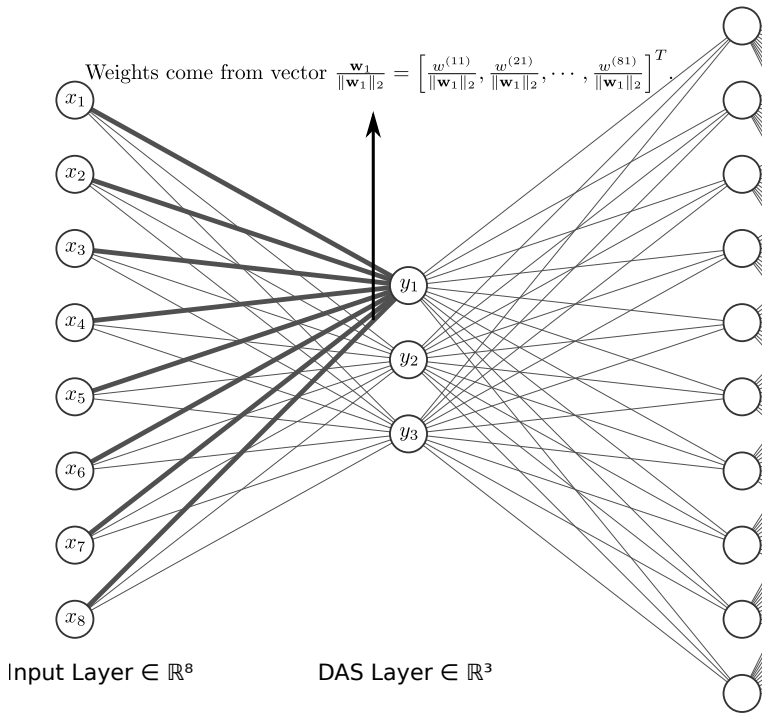


Figure 1. *Diagram showing a DAS layer with $D = 8$ and $d = 3$. The thick lines contain the weights coming from the first column vector of $W_1(Q)$. Likewise, the second column vector contains the weights of all lines ending at neuron y_2, etc.*

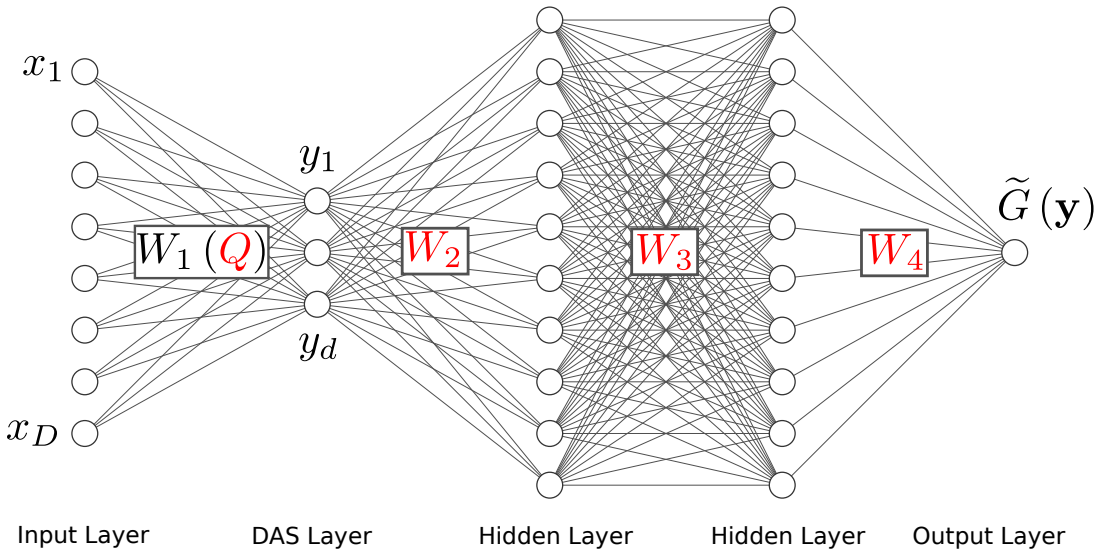


Figure 2. *A full schematic of a DAS surrogate. The red matrices (Q, W_2, W_3, and W_4) are optimized using stochastic gradient descent and back propagation.*

algorithm [1], in which the gradient $\partial L/\partial W_i$ is computed (where $L \in \mathbb{R}$ is the loss function). Generally, the weights are updated via $W_i = W_i - \alpha \partial L/\partial W_i$, where α is the learning rate, specified later.

The situation in the DAS layer is different. Since $W_1 = W_1(Q)$, we need to optimize the loss with respect to Q instead of directly optimizing W_1. Hence, we also require $\partial L/\partial Q$, which in turn, via the chain rule, requires the derivates of $\mathbf{w}_i/\|\mathbf{w}_i\|_2$ with respect to the $\mathbf{q}_k$ vectors. The authors of [37] suggest using automatic differentiation. This does make sense, since although $\mathbf{w}_i/\|\mathbf{w}_i\|_2$ is algebraic and differentiable, it quickly becomes a complicated expression involving a very large number of q_{ij} terms. Here, q_{ij} are the entries of $\mathbf{q}_j = [q_{1j}, \ldots, q_{Dj}]^T$, $j = 1, \ldots, d$. That said, we will show that we can also use matrix calculus to find a simple analytic expression for $\partial L/\partial Q$. Note that this is an alternative to the approach of [37], where Gram–Schmidt was implemented in an automatic-differentiation capable library such as PyTorch [36]. The main reason we use the analytic expression here, is that these are new expressions which might be useful outside a machine-learning context as well. Second, the analytic expressions are intuitive, as we can see that the Gram–Schmidt derivatives are a simple sum of matrix-matrix multiplications (see Appendix A). This gives insight into the structure and cost of computing $\partial L/\partial Q$, as we also derive an analytic formula for the number of matrix-matrix multiplications involved. We do not compare the performance of our code to the automatic differentiation of, e.g., PyTorch in terms of training time. The largest training time we observed was around 30 seconds, and even if we could exploit some (potential) methodological advantage with the analytic expressions, in practice it is unlikely that our in-house Python implementation (section 8.3) will outperform the highly optimized C++ backend of PyTorch.

3.1. Differentiating Gram–Schmidt. The gradient matrix of L with respect to Q is given by

$$\frac{\partial L}{\partial Q} = \begin{bmatrix} \frac{\partial L}{\partial q_{11}} & \cdots & \frac{\partial L}{\partial q_{1d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial q_{D1}} & \cdots & \frac{\partial L}{\partial q_{Dd}} \end{bmatrix} \in \mathbb{R}^{D\times d}, \tag{3.3}$$

where each entry is computed via the chain rule as

$$\frac{\partial L}{\partial q_{ij}} = \frac{\partial L}{\partial w_{11}}\frac{\partial w_{11}}{\partial q_{ij}} + \frac{\partial L}{\partial w_{12}}\frac{\partial w_{12}}{\partial q_{ij}} + \cdots + \frac{\partial L}{\partial w_{Dd}}\frac{\partial w_{Dd}}{\partial q_{ij}}, \quad i = 1,\dots,D, \quad j = 1,\dots,d. \tag{3.4}$$

This expansion contains Dd terms, although $\partial w_{kl}/\partial q_{ij} = 0$ whenever $j > l$. This can be seen by examining (3.2), in which the dependence of the $\mathbf{w}$ vectors on the $\mathbf{q}$ vectors is made explicit. The terms of the summation (3.4) are those resulting from the elementwise multiplication of the matrices $\partial L/\partial W_1$ and $\partial W_1/\partial q_{ij}$. Thus, we can rewrite (3.4) in shorthand as

$$\frac{\partial L}{\partial q_{ij}} = \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{ij}} \right\rangle_F, \tag{3.5}$$

where $\langle\cdot,\cdot\rangle_F$ is the Frobenius inner product. Thus, the gradient we must compute to back propagate the loss through the DAS layer is given by

$$\frac{\partial L}{\partial Q} = \begin{bmatrix} \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{11}} \right\rangle_F & \cdots & \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{1d}} \right\rangle_F \\ \vdots & \ddots & \vdots \\ \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{D1}} \right\rangle_F & \cdots & \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{Dd}} \right\rangle_F \end{bmatrix}. \tag{3.6}$$

Note that $\partial L/\partial W_1$ will be available through standard back propagation. However, the Dd matrices $\partial W_1/\partial q_{ij}$ will require us to compute the derivatives of the Gram–Schmidt vectors $\mathbf{w}_i$ with respect to the original, nonorthogonal vectors $\mathbf{q}_k$.

3.1.1. Derivatives of unnormalized Gram-Schmidt vectors. The derivatives of $\mathbf{w}_i$ with respect to $\mathbf{q}_k$ are given by the following recurrence relationships:

$$\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_1} =: D_{11} = I_D, \tag{3.7}$$

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_i} =: D_{ii} = D_{i-1,i-1} - \frac{\mathbf{w}_{i-1}\mathbf{w}_{i-1}^T}{\mathbf{w}_{i-1}^T\mathbf{w}_{i-1}}, \quad i > 1, \tag{3.8}$$

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_k} = \sum_{j=1}^{i-1} D_{ij}\frac{\partial \mathbf{w}_j}{\partial \mathbf{q}_k}, \quad i \neq k, \quad i > k, \tag{3.9}$$

$$\begin{aligned} D_{ij} &:= -\frac{\partial}{\partial \mathbf{w}_j}\left[\left(\frac{\mathbf{w}_j^T\mathbf{q}_i}{\mathbf{w}_j^T\mathbf{w}_j}\right)\mathbf{w}_j\right] \\ &= -\left[\frac{1}{\mathbf{w}_j^T\mathbf{w}_j}\,\mathbf{w}_j\mathbf{q}_i^T - \frac{2\mathbf{w}_j^T\mathbf{q}_i}{(\mathbf{w}_j^T\mathbf{w}_j)^2}\,\mathbf{w}_j\mathbf{w}_j^T + \frac{\mathbf{w}_j^T\mathbf{q}_i}{\mathbf{w}_j^T\mathbf{w}_j}\,I_D\right], \quad i \neq j, \quad i > j. \end{aligned} \tag{3.10}$$

These are derived in Appendix A. Note that we have separate expressions for the "normal" derivatives $\partial \mathbf{w}_i/\partial \mathbf{q}_i$ and "shear" derivatives $\partial \mathbf{w}_i/\partial \mathbf{q}_j$ (where $i \neq j$), i.e., (3.8) and (3.9), respectively. The latter can be expanded as a sum of matrix-matrix multiplications involving only D_{ij} matrices (see Appendix A), although in practice we will compute (3.8) and (3.9) recursively, starting with $i = 1$.

3.1.2. Derivatives of normalized Gram–Schmidt vectors. Equations (3.9) and (3.8) give simple expressions for $\partial \mathbf{w}_i/\partial \mathbf{q}_k$. However, the weight vectors of the DAS layers are normalized (see Figure 1), and so we need to compute $\partial(\mathbf{w}_i/\|\mathbf{w}_i\|_2)/\partial \mathbf{q}_k$. As shown in Appendix B, we can just premultiply $\partial \mathbf{w}_i/\partial \mathbf{q}_k$ with a matrix which only depends upon $\mathbf{w}_i$, to obtain the gradient of the corresponding normed vector:

$$\frac{\partial}{\partial \mathbf{q}_k}\left(\frac{\mathbf{w}_i}{\|\mathbf{w}_i\|_2}\right) = \left[\frac{I_D}{\|\mathbf{w}_i\|_2} - \frac{\mathbf{w}_i \mathbf{w}_i^T}{\|\mathbf{w}_i\|_2^3}\right] \frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_k}. \tag{3.11}$$

We have placed our Python subroutines, as well as symbolic math scripts to verify the validity of (3.7)–(3.11) on a separate GitHub repository; see [14].

3.1.3. Assembling the loss gradient. Computing all $\partial(\mathbf{w}_i/\|\mathbf{w}_i\|_2)/\partial \mathbf{q}_k$ gives us the information we need to assemble the loss gradient (3.6), provided that standard back propagation has provided $\partial L/\partial W_1$. However, the information is not yet in the right place to compute the Frobenius inner products $\langle \partial L/\partial W_1, \partial W_1/\partial q_{ij} \rangle_F$ of (3.6), because these require $\partial W_1/\partial q_{ij}$ instead of $\partial \mathbf{w}_i/\partial \mathbf{q}_k$. The matrices $\partial \mathbf{w}_i/\partial \mathbf{q}_k$ and $\partial W_1/\partial q_{ij}$ are given by

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_k} = \begin{bmatrix} \frac{\partial w_{1i}}{\partial q_{1k}} & \frac{\partial w_{1i}}{\partial q_{2k}} & \cdots, & \frac{\partial w_{1i}}{\partial q_{Dk}} \\ \frac{\partial w_{2i}}{\partial q_{1k}} & \frac{\partial w_{2i}}{\partial q_{2k}} & \cdots, & \frac{\partial w_{2i}}{\partial q_{Dk}} \\ \vdots & & \ddots & \vdots \\ \frac{\partial w_{Di}}{\partial q_{1k}} & \frac{\partial w_{Di}}{\partial q_{2k}} & \cdots, & \frac{\partial w_{Di}}{\partial q_{Dk}} \end{bmatrix} \in \mathbb{R}^{D\times D}, \quad \frac{\partial W_1}{\partial q_{ij}} = \begin{bmatrix} \frac{\partial w_{11}}{\partial q_{ij}} & \frac{\partial w_{12}}{\partial q_{ij}} & \cdots, & \frac{\partial w_{1d}}{\partial q_{ij}} \\ \frac{\partial w_{21}}{\partial q_{ij}} & \frac{\partial w_{22}}{\partial q_{ij}} & \cdots, & \frac{\partial w_{2d}}{\partial q_{ij}} \\ \vdots & & \ddots & \vdots \\ \frac{\partial w_{D1}}{\partial q_{ij}} & \frac{\partial w_{D2}}{\partial q_{ij}} & \cdots, & \frac{\partial w_{Dd}}{\partial q_{ij}} \end{bmatrix} \in \mathbb{R}^{D\times d}. \tag{3.12}$$

To compute the Frobenius inner products $\langle \partial L/\partial W_1, \partial W_1/\partial q_{ij} \rangle_F$, we need to assemble Dd matrices $\partial W_1/\partial q_{ij}$. From (3.12), we see that this is done as

$$\frac{\partial W_1}{\partial q_{ij}} = \left[\operatorname{col}\left(i, \frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_j}\right) \ \operatorname{col}\left(i, \frac{\partial \mathbf{w}_2}{\partial \mathbf{q}_j}\right) \ \cdots \ \operatorname{col}\left(i, \frac{\partial \mathbf{w}_d}{\partial \mathbf{q}_j}\right) \ \right], \tag{3.13}$$

where $\operatorname{col}(i, A)$ returns the ith column of matrix A. Hence, we can easily assemble $\partial W_1/\partial q_{ij}$ by selecting the ith column of the d derivatives $\partial \mathbf{w}_k/\partial \mathbf{q}_j$, $k = 1, \ldots, d$. In fact, if we store all $\partial \mathbf{w}_i/\partial \mathbf{q}_k$ derivatives in a 3-dimensional (3D) array of shape $[d^2, D, D]$, we can form the $\partial W_1/\partial q_{ij}$ matrices by just taking 2-dimensional (2D) slices from this array.

4. Orthonormality through postprocessing. The DAS method as described previously always maintains orthonormality of W_1, also during training. However, as pointed out by a reviewer, this may not be necessary. Consider a feed-forward neural network where the weight matrices are denoted by M_i. The architecture of this network is the same as the DAS network shown in Figure 2, except orthonormality is not enforced for $M_1 \in \mathbb{R}^{D\times d}$, i.e., $M_1 \neq M_1(Q)$.

To differentiate this network from the DAS network, we denote it as a "constrained" ANN, due to the fact that the number of neurons in the first hidden layer is constrained to the dimension of the active subspace d. In order to maintain a connection to the original active-subspace method (which serves as a reference throughout this article), we must now extract an orthonormal projection matrix a posteriori, i.e., after training. Specifically, if we apply the *original* active-subspace method to the *surrogate* $\widetilde{f}$, the resulting dominant eigenvectors of the neural-network equivalent of C (denoted by V_1), can perform the same function as $W_1(Q)$.

4.1. Derivative of a neural network. To compute the original active subspace of (any) neural network $\widetilde{f}$, we must compute $\partial\widetilde{f}/\partial\mathbf{x}$. Let $M_r \in \mathbb{R}^{p_{r-1}\times p_r}$ be the weight matrix of the rth layer, for $r = 0, 2, \ldots, N$. Here, p_{r-1} is the number of neurons of the preceding layer, and p_r is the number of neurons of the rth layer (not including the bias neuron). Each layer, except the input layer ($r = 0$), has its own weight matrix M_r. Let $\mathbf{h}_i := \Phi(M_r^T\mathbf{h}_{r-1}) = \Phi(\mathbf{a}_r) \in \mathbb{R}^{p_r}$ be the activation of the rth layer, with activation function $\Phi(\cdot)$. Finally, let $D_r := \mathrm{diag}(\Phi'(\mathbf{a}_r)) \in \mathbb{R}^{p_r\times p_r}$ be the diagonal matrix with $\Phi'(a_r^{(i)}) := \mathrm{d}\Phi(a_r^{(i)})/\mathrm{d}a_r^{(i)}$ as entries, for $i = 1, \ldots, p_r$. If the output is scalar (or if we are computing the gradient of a norm of the output vector), we can compute $\partial\widetilde{f}/\partial\mathbf{x}$ with just a small modification of the standard back propagation algorithm. Working from the output layer backwards, we have the following recurrence relation,

$$\frac{\partial\widetilde{f}}{\partial h_N} = \frac{\partial\widetilde{f}}{\partial\widetilde{f}} = 1, \quad r = N, \tag{4.1}$$

$$\frac{\partial\widetilde{f}}{\partial\mathbf{h}_r} = M_{r+1}D_{r+1}\frac{\partial\widetilde{f}}{\partial\mathbf{h}_{r+1}}, \quad r = N-1, N-2, \ldots, 0; \tag{4.2}$$

see, e.g., [12 1]. When we set $\widetilde{f} = L$, we obtain the standard back propagation algorithm for computing $\partial L/\partial\mathbf{h}_r$. In (4.1) we assume that the activation h_N of the output layer equals $\widetilde{f}$. If standardization is used during training (as is common practice), $h_N = (\widetilde{f} - \mu_f)/\sigma_f$, where μ_f and σ_f are the data mean and standard deviation. Following (4.2), at $r = 0$ we get

$$\frac{\partial\widetilde{f}}{\partial\mathbf{x}} = M_1D_1\frac{\partial\widetilde{f}}{\partial\mathbf{y}} \in \mathbb{R}^{D\times 1}. \tag{4.3}$$

Then,

$$\left(\frac{\partial\widetilde{f}}{\partial\mathbf{x}}\right)\left(\frac{\partial\widetilde{f}}{\partial\mathbf{x}}\right)^T = M_1D_1\left(\frac{\partial\widetilde{f}}{\partial\mathbf{y}}\right)\left(\frac{\partial\widetilde{f}}{\partial\mathbf{y}}\right)^T D_1M_1^T.$$

Let C_{ANN} be the neural-network equivalent of C, i.e.,

$$C_{ANN} = \int\left(\frac{\partial\widetilde{f}}{\partial\mathbf{x}}\right)\left(\frac{\partial\widetilde{f}}{\partial\mathbf{x}}\right)^T p(\mathbf{x})d\mathbf{x} = M_1\int D_1\left(\frac{\partial\widetilde{f}}{\partial\mathbf{y}}\right)\left(\frac{\partial\widetilde{f}}{\partial\mathbf{y}}\right)^T D_1\, p(\mathbf{x})d\mathbf{x}\, M_1^T. \tag{4.4}$$

Since the network is fast to evaluate, we can easily approximate this via MC sampling:

$$C_{ANN} \approx \overline{C}_{ANN} = \overline{\left(\frac{\partial\widetilde{f}}{\partial\mathbf{x}}\right)\left(\frac{\partial\widetilde{f}}{\partial\mathbf{x}}\right)^T} = M_1D_1\overline{\left(\frac{\partial\widetilde{f}}{\partial\mathbf{y}}\right)\left(\frac{\partial\widetilde{f}}{\partial\mathbf{y}}\right)^T}D_1M_1^T, \tag{4.5}$$

where $\overline{X}$ denotes the MC approximation of X. Following the original active-subspace method, we can orthogonally diagonalize $\overline{C}$ since it is a symmetric matrix,

$$\overline{C}_{ANN} = [V_1 \;\; V_2] \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} [V_1 \;\; V_2]^T = M_1 \overline{C}_1 M_1^T, \tag{4.6}$$

where we have introduced the shorthand $\overline{C}_1 := \overline{D_1 \left(\partial \widetilde{f}/\partial \mathbf{y}\right) \left(\partial \widetilde{f}/\partial \mathbf{y}\right)^T D_1} \in \mathbb{R}^{d\times d}$. Note that in the DAS case we have $D_1 = I_d$, since there the activation is linear in the first layer.

4.2. Eigendecomposition of $\overline{C}_{ANN}$. We state the following lemma.

Lemma 4.1. *Any neural network, with d neurons in the first hidden layer, has a $\overline{C}_{ANN}$ gradient matrix with at most d nonzero eigenvalues, such that $\Lambda_2 = \mathbf{0}$.*

Proof. $\overline{C}_1$ is symmetric positive semidefinite, since

$$\mathbf{a}^T \overline{C}_1 \mathbf{a} = \frac{1}{I} \sum_{i=1}^{I} \mathbf{a}^T D_{1i} \left(\frac{\partial \widetilde{f}}{\partial \mathbf{y}_i}\right) \left(\frac{\partial \widetilde{f}}{\partial \mathbf{y}_i}\right)^T$$
$$D_{1,i}\mathbf{a} = \frac{1}{I} \sum_{i=1}^{I} \left(\frac{\partial \widetilde{f}}{\partial \mathbf{y}_i}^T D_{1,i}\mathbf{a}\right)^T \left(\frac{\partial \widetilde{f}}{\partial \mathbf{y}_i}^T D_{1,i}\mathbf{a}\right) \geq 0 \;\; \forall \mathbf{a} \in \mathbb{R}^d.$$

Therefore $\overline{C}_1 \in \mathbb{R}^{d\times d}$ has d real eigenvalues $\mu_i \geq 0$, and $\operatorname{rank}(\overline{C}_1) \leq d$, with equality when all $\mu_i > 0$. Since $M_1 \in \mathbb{R}^{D\times d}$, $\operatorname{rank}(M_1) \leq d$. From standard textbooks on linear algebra we know that $\operatorname{rank}(AB) \leq \min\{\operatorname{rank}(A), \operatorname{rank}(B)\}$, and $\operatorname{rank}(\mathrm{A}) = \operatorname{rank}(\mathrm{A}^{\mathrm{T}})$. Hence $\operatorname{rank}(\overline{\mathrm{C}}_{\mathrm{ANN}}) = \operatorname{rank}(M_1 \overline{C}_1 M_1^T) \leq d$. Let Λ_1 contain the d largest eigenvalues. Since $D = \operatorname{rank}(\overline{C}_{ANN}) + \dim(\operatorname{null}(\overline{C}_{ANN}))$, the null space of $\overline{C}_{ANN}$ is at least $D - d$ dimensional, and hence $\Lambda_2 = \mathbf{0}$. In active-subspace terminology, we say that the network is $\mathbf{z}$-invariant. ∎

This also holds for the DAS network, which makes intuitive sense, since it has no representation of the inactive variables $\mathbf{z} = V_2^T \mathbf{x} \in \mathbb{R}^{D-d}$ in the DAS layer. Since $\Lambda_2 = \mathbf{0}$, we can rewrite (4.6) as

$$\overline{C}_{ANN} = V_1 \Lambda_1 V_1^T = M_1 \overline{C_1} M_1^T \in \mathbb{R}^{D\times D}. \tag{4.7}$$

Clearly, once we solve the eigendecomposition of $\overline{C}_{ANN}$, we can use V_1 in a way similar to W_1, i.e., to find the active subspace as $\mathbf{y} = V_1^T \mathbf{x}$. As a side note, in the case of a DAS network we write C_{DAS}. Here we can solve the smaller eigendecomposition of $\overline{C}_1$ to obtain both Λ_1 and V_1 (making use of the orthonormal nature of $W_1(Q)$). $\overline{C}_1$ shares the Λ_1 eigenvalues of $\overline{C}_{DAS}$, and V_1 is found by premultiplying the $\overline{C}_1$ eigenvectors with W_1. However, since we already found W_1 during training, we have no need for a second projection matrix.

5. Vector-valued QoIs. The construction of C assumes that the QoI f is scalar; see (2.1). That said, recent work has shown that with some modification, the active-subspace method can be extended to vector-valued QoIs. These are based on minimizing an upper bound on the approximation error [39] or modeling the output with a truncated Karhunen–Loève expansion [23]. The constrained ANN and DAS method can handle vector-valued QoI

without modification, as it will just increase the number of output neurons, without changing the DAS layer. However, by this we mean that we can train a surrogate on vector-valued outputs. Whether or not the network converges to a meaningful active subspace depends on the underlying assumption, that (besides the existence of a linear active subspace), the different outputs $f_i \in \mathbf{f}$ also share approximately the same active subspace, which could be unrealistic.

After training, we can compare our surrogates to the reference active subspace of $\|\mathbf{f}\|_2$, which is something that can be computed using the original active-subspace method if $\partial\|\mathbf{f}\|_2/\partial\mathbf{x}$ is available. The neural-network equivalent of (2.1) in this case is

$$\overline{C}_{ANN} = \overline{\left(\frac{\partial\|\widetilde{\mathbf{f}}\|_2}{\partial\mathbf{x}}\right)\left(\frac{\partial\|\widetilde{\mathbf{f}}\|_2}{\partial\mathbf{x}}\right)^T}. \tag{5.1}$$

The gradient appearing in (5.1) can still be computed exactly, by replacing (4.1) with $\partial\|\widetilde{\mathbf{f}}\|_2/\partial\mathbf{h}_N = \partial\|\widetilde{\mathbf{h}}_N\|_2/\partial\mathbf{h}_N = \mathbf{h}_N/\|\mathbf{h}_N\|_2$, and proceeding from there in the same manner as for the computation of $\partial\widetilde{f}/\partial\mathbf{x}$.

6. Sensitivity analysis. Besides uncertainty propagation, another common use case of uncertainty quantification is to assess which inputs are more influential than others. One of the most common options is to use global variance-based sensitivity methods (e.g., [34]). Other global approaches are derivative based, e.g.,

$$\nu_i := \int \left(\frac{\partial f}{\partial x_i}\right)^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{6.1}$$

These indices measure the (average) sensitivity of f to small perturbations in the inputs $\mathbf{x}$, and are especially suited for identifying noninfluential parameters [35]. Note that the ν_i indices are the diagonal elements of the matrix C, such that

$$\begin{aligned}[\nu_1,\dots,\nu_D]^T &= \mathrm{diag}(C) = \mathrm{diag}\left(U\Lambda U^T\right)\\ &= \left[\sum_{j=1}^{D}\lambda_j U_{1j}^2,\dots,\sum_{j=1}^{D}\lambda_j U_{Dj}^2\right]^T =: [\alpha_1(D),\dots,\alpha_D(D)]^T.\end{aligned} \tag{6.2}$$

The α_i are called "activity scores" [6] in the context of active subspaces, and from (6.2) we see that $\alpha_i(D) = \nu_i$. That said, active subspaces are truncated at some $d < D$, such that

$$\alpha_i(d) := \sum_{j=1}^{d}\lambda_j U_{ij}^2 \leq \alpha_i(D) = \nu_i. \tag{6.3}$$

Since we can compute the exact $\partial\widetilde{f}/\partial x_i$ of a neural network very quickly, an MC approximation of (6.1), replacing $\partial f/\partial x_i$ by $\partial\widetilde{f}/\partial x_i$, is readily obtained. For the constrained ANN and DAS surrogate we further know that $\widetilde{f}$ is $\mathbf{z}$-invariant, such that $\lambda_i = 0$ for $i > d$. In this case (6.3) becomes $\alpha_i(d) = \alpha_i(D) = \nu_i$,, i.e., also the truncated activity scores $\alpha_i(d)$ are exactly the same as the global derivative-based sensitivity measures.

7. Other derivative-free active-subspace approaches. For comparison purposes we briefly outline one other simple alternative for derivative-free active-subspace estimation. In particular, we focus on the use of linear regression in a "quick-and-dirty check" [4] for the existence of a one-dimensional (1D) active subspace. Briefly, consider the linear-regression model

$$f(\mathbf{x}) \approx \widetilde{f}(\mathbf{x}) = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_D x_D, \tag{7.1}$$

where the coefficients a_i are estimated via standard least squares; see [4] for more details. Clearly, the derivative of the surrogate $\widetilde{f}$ is given by $\mathbf{a}' := [a_1, a_2, \ldots, a_D]^T$. While the linear regression surrogate may be inaccurate, if a single dominant and monotonic trend is present in f, this simple method may still be used to identify this trend. In other words, if f contains a 1D active subspace, $\mathbf{a}'/\|\mathbf{a}'\|_2$ may serve to approximate it.

8. Results. Let us now show the results of all the preceding analyses. Our applications involve two disease models, namely, an HIV model and a much more computationally demanding COVID19 model. Three network types will be considered. The DAS network from section 3, the constrained ANN from section 4, and a standard (unconstrained) ANN, where the number of neurons in the first hidden layer is not constrained to the dimension of the active subspace. The linear regression approach is tested on the HIV model only.

8.1. HIV model. The original active-subspace method has already been applied to this model [29], and the source code and data are available from [5]. This provides us with a reference solution, where the derivatives needed to build the $\overline{C}$ matrix are finite-difference approximations. Briefly, the model consists of 7 coupled ordinary differential equations (ODEs),

$$\begin{aligned}
\frac{dT}{dt} &= s_1 + \frac{p_1}{C_1 + V} TV - \delta_1 T - (K_1 V + K_2 M_I)T, \\
\frac{dT_I}{dt} &= \psi(K_1 V + K_2 M_I)T + \alpha_1 T_L - \delta_2 T_I - K_3 T_I CTL, \\
\frac{dT_L}{dt} &= (1-\psi)(K_1 V + K_2 M_I)T - \alpha_1 T_L - \delta_3 T_L, \\
\frac{dM}{dt} &= s_2 + K_4 MV - K_5 MV - \delta_4 M, \\
\frac{dM_I}{dt} &= K_5 MV - \delta_5 M_I - K_6 M_I CTL, \\
\frac{dCTL}{dt} &= s_3 + (K_7 T_I + K_8 M_I)CTL - \delta_6 CTL, \\
\frac{dV}{dt} &= K_9 T_I + K_{10} M_I - K_{11} TV - (K_{12} + K_{13})MV - \delta_7 V,
\end{aligned} \tag{8.1}$$

with 27 input parameters that we model via uniform distributions with boundaries set at ± 2.5% of their nominal values (see the supplementary materials (supplement.pdf [local/web 184KB]) for specific values, and for a description of the ODE variables). We just note that our QoI is the T-cell count $T(t)$ over time, and we refer to [29] for further information on the model.

We will consider two cases, namely, one that is intentionally overfitted (using 100 neurons per hidden layer), and one that is not, with 10 neurons per hidden layer. Rather than finding

the best model via an exhaustive hyperparameter search, our goal here is to contrast the performance of the different networks in these two cases. For all networks the standard squared loss function, $\Phi = tanh$ activation, and a minibatch size of 64 is used. Back propagation is performed with stochastic gradient descent with the RMSProp optimizer and a learning rate of 0.001 [1]. The number of training iterations is set to 10000. To make sure that all inputs have a common scale, we normalize the inputs such that $x_i \in [-1, 1]$ for all $i = 1, \ldots, D$, which is common in the uncertainty-quantification literature. For the T-cell count output data we experimented with normalization to the unit interval $[0, 1]$, and with standardization to mean 0 and standard deviation 1. We found that the latter technique yielded more accurate surrogates. Hence, in the following, consider f as standardized unless noted otherwise, and $\widetilde{f}$ as the corresponding surrogate.

The reference eigenvalues are shown in Figure 3. There is a large gap between λ_1 and λ_2, indicating the existence of a 1D active subspace [9]. Clearly, we cannot rely on such information to be available in general, and we therefore need a strategy of determining d. A simple method consist of computing the eigenvalues Λ_1 of $\overline{C}_{ANN}$ or $\overline{C}_{DAS}$ for some $d > 1$, and to look for a large eigenvalue gap there. Figure 3 also displays the constrained ANN and DAS surrogate eigenvalues with $d = 5$, and in both cases we also observe a large gap between λ_1 and λ_2. Hence, even without the reference solution we would come to the same conclusion, i.e., in all subsequent analyses we set $d = 1$. Figure 3 also shows that $\Lambda_2 = 0$ as discussed in section 4.2.

Figure 4 shows a heat map of the $\overline{C}$ matrices, for the reference active-subspace method computed with finite-difference gradients of the code, and the same heat map for the constrained ANN and DAS surrogate, for which no direct gradient data were used. The scalar QoI $f(\mathbf{x})$ was the T-cell count at $t = 45$ days. Both the ANN and DAS surrogates capture the overall structure of $\overline{C}$ quite well compared to the reference case, despite the lack of gradient data.

Next we examine the relative training and test error distribution, as a function of the training data size. Both the constrained ANN and DAS networks have $d = 1$, such that the main difference between them is the imposed orthogonality of W_1 in the architecture of the

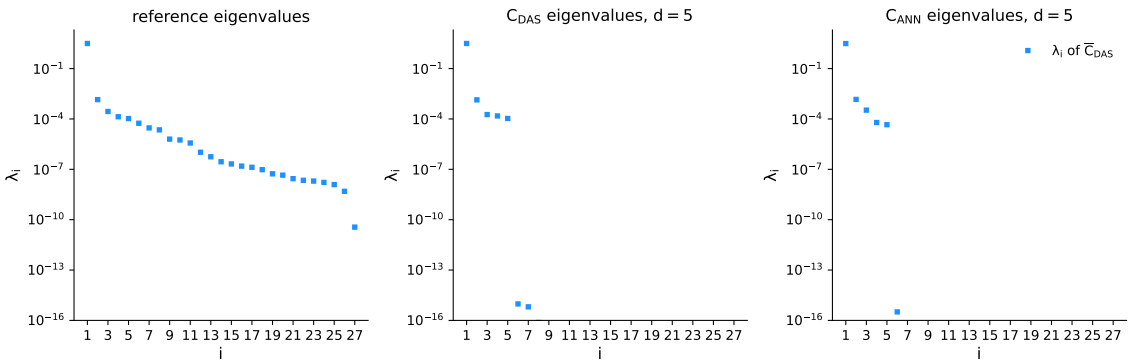


Figure 3. *Left: the reference eigenvalues of $\overline{C}$. Right: the eigenvalues of $\overline{C}_{ANN}$ in the case of a constrained ANN. Middle: the eigenvalues of the $\overline{C}_{DAS}$ matrix. Due to Lemma 4.1, the eigenvalues λ_i are (numerically) zero for $\overline{C}_{ANN}$ and $\overline{C}_{DAS}$ when $i > d$.*

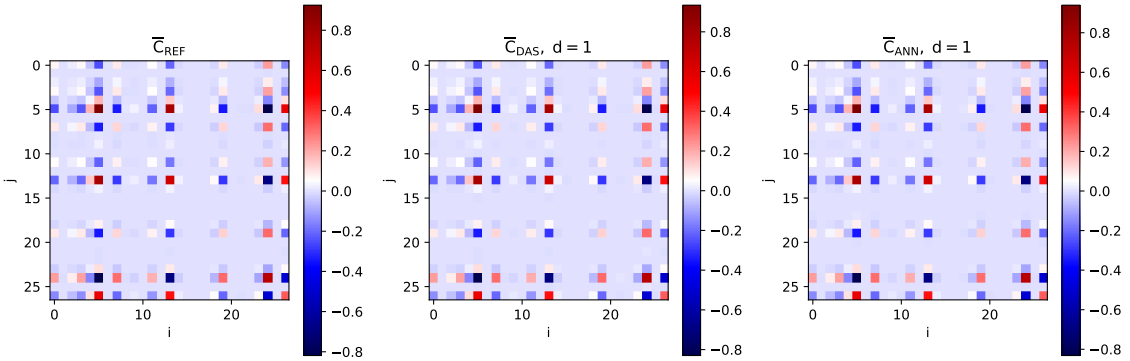


Figure 4. *Heat map of the gradient matrix* $\overline{C} = \overline{(\partial f/\partial \mathbf{x})(\partial f/\partial \mathbf{x})^T}$ *for the reference active-space method, and the corresponding DAS and constrained ANN approximations based on* $\partial \widetilde{f}/\partial \mathbf{x}$. *The outputs* f *and* $\widetilde{f}$ *were standardized.*

DAS network. To isolate the effect of this orthonormality we also train an unconstrained ANN. Due to the stochastic nature of the training, the error is a random variable. We therefore train 100 replica networks, for every network type and at every training data size to compute 95% confidence intervals of the errors. We define the relative error as a percentage,

$$e = \frac{1}{T}\sum_{i=1}^{T} \frac{|f(\mathbf{x}_i) - \widetilde{f}(\mathbf{x}_i)|}{|f(\mathbf{x}_i)|} \times 100. \tag{8.2}$$

Here, T is either the size of the training or test data set. We have 1000 code samples in total, and consider training data sizes ranging from 500 to 900. The test data are whatever remains from the total data set. The results for the overfitted case are shown in Figure 5. Note that all displayed errors are low, less than 0.5% in all cases. That said, the confidence intervals show that variation of the unconstrained ANN error is significantly larger than that of the other two surrogates. And while the unconstrained ANN achieves a lower training error on average, a relatively large jump in its test error is observed, which progressively decreases with more training data. No discernable difference between training and test performance can be seen for the DAS surrogate or the constrained ANN, and the general trend of both the training and test error is constant with respect to the considered range of training data sizes. Further note that the error confidence intervals of the DAS surrogate and the constrained ANN are very similar. This shows that funneling the inputs through a hidden layer constrained to just d neurons is what causes the good generalization properties, rather than the orthonormality of W_1. Note that since $d = 1$ here, $\mathbf{w}_1$ is only orthonormal with respect to itself, but the confidence intervals were also very similar for $d = 2$ (data available at [15]).

The jump in the unconstrained ANN test error is a clear indication of overfitting. Figure 6 shows the same results when using 10 (instead of 100) neurons per hidden layer. Again, all errors are very low, yet now the unconstrained ANN generally gives a lower test error than the other surrogates. That said, the noise in the unconstrained ANN error is still relatively large, and the DAS network and constrained ANN are still more robust in the sense that not much difference can be observed between the training and test performance. While we could have used, for instance, L2 regularization to reduce the unconstrained ANN overfitting, these

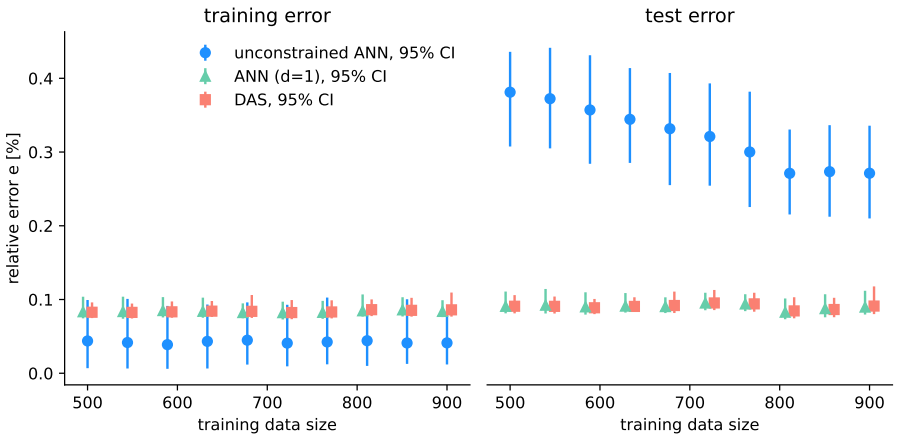


Figure 5. *The training (left) and test (right) error for the DAS and (constrained and unconstrained) ANN surrogates versus the training data size, using* 100 *neurons for the hidden layers. The* 95% *confidence intervals are computed from* 100 *replica neural networks, independently trained at every training data size. Since we have* 10 *training data sizes and* 3 *surrogate methods, a total of* 3000 *neural networks are trained to generate these results.*

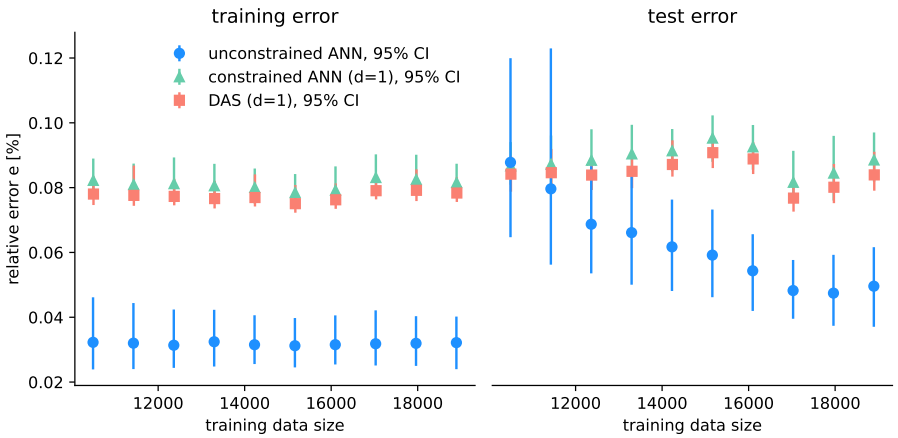


Figure 6. *The training (left) and test (right)* 95% *error confidence intervals for the DAS and (constrained and unconstrained) ANN surrogates versus the training data size, using* 10 *neurons for the hidden layers.*

results indicate that restricting the number of neurons to the dimension of the active subspace also can act as a means of regularization.

To see why, consider Figure 7, which shows the predicted response in the active subspace of the DAS and ANN surrogates (i.e., $\widetilde{G}(y_1)$), at $t = 3400$ days. As a validation exercise, we also plot 100 new random code evaluations along y_1. Clearly, the model outcome is very well predicted along this dimension in the case of the DAS and constrained ANN surrogates. Since the active subspace of the HIV model is (approximately) 1D, and we also restrict the $\mathbf{y}$ space of these surrogates to one dimension, we essentially fit the "signal," while ignoring the "noise." The unconstrained ANN does fit the noise; again see Figure 7. This may result in a smaller training error, but may not generalize well and is likely to cause more variation between replica networks.

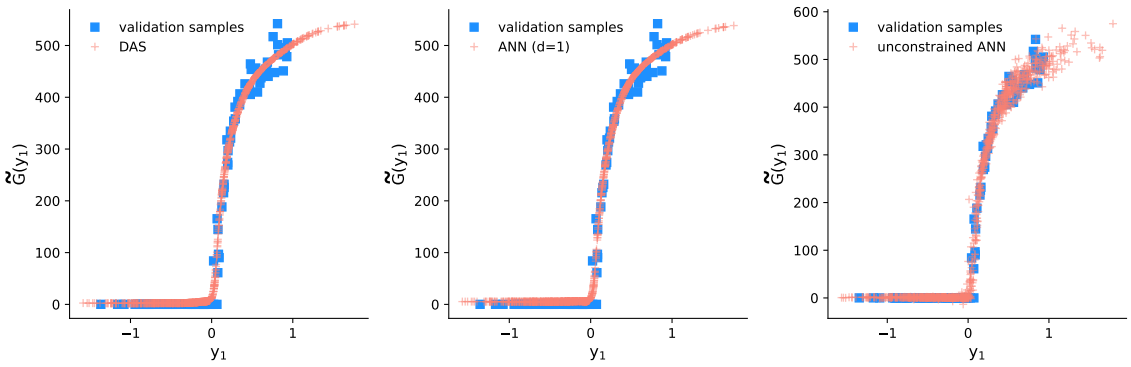


Figure 7. *The predictions at* $t = 3400$ *days, and* 100 *(nonstandardized) validation samples* $f(\mathbf{x}_i^*)$, $i = 1, \ldots, 100$, *from* (8.1), *plotted along the* y_1 *coordinate extracted from the DAS (right), constrained ANN (middle), and unconstrained ANN (right). All networks used* 100 *neurons per hidden layer, besides the DAS and constrained ANN obviously, which used* $d = 1$ *neuron in the first hidden layer.*

Hence, even though the model has 27 inputs, it is (almost) a 1D function if one is able to identify the active subspace. These results indicate that $\mathbf{u}_1 \approx \mathbf{w}_1/\|\mathbf{w}_1\|_2$ (up to multiplication by -1), i.e., that the 1D active subspace spanned by the dominant orthonormal eigenvector of U approximately equals the (only) column vector of W_1. Likewise, we have $\mathbf{u}_1 \approx \mathbf{v}_1$. For $d > 1$ the DAS coordinate system will likely differ from the reference active subspace by some arbitrary rotation, as also observed by [37]. That said, we only judge the performance of the surrogate by whether or not it (approximately) identifies the same subspace as the original active-subspace method, e.g., if $\text{span}\,(U_1) \approx \text{span}(W_1)$ or $\text{span}\,(U_1) \approx \text{span}(V_1)$ holds or not. Since it does hold for the HIV model, we could also use a more classical (regression-based) surrogate method to construct $\widetilde{G}(y_1)$. In this case we would not use the neural network $\widetilde{G}(y_1)$ for prediction, and instead only use the network to generate a projection matrix W_1 or V_1, and subsequently train our surrogate of choice in the corresponding active subspace.

Taking this argument one step further, in section 7 we discussed a linear-regression based method to identify a 1D active subspace, cutting out machine learning altogether. The results of this method are shown in Figure 8. While linear regression is clearly too simplistic as a surrogate, the validation samples are comparable to those in Figure 7. Hence, this simple (derivative-free) method is able to identify the 1D active subspace here. The advantage of machine-learning based approaches is that they simultaneously act as a powerful surrogate modelling method, and they also provide actual benefit when the active-subspace dimension is 2 or higher.

Next we plot the global gradient-based sensitivity indices ν_i, $i = 1, \ldots, 27$, in Figure 9, for both the reference active subspace, DAS and constrained ANN surrogates at $t = 3400$ days. While the magnitude is not an exact match, the order is exactly the same for all parameters that have a significant nonzero ν_i. That said, if we retrain the surrogates, parameters that are very close to each other (such as s_2 and K_7), might switch places. Hence, here it may also be useful to consider replica networks. In any case, while results such as these are useful to eliminate a lot of individual parameters, they do not show that here we can collapse the input dimensionality further to 1D by taking a linear combination of parameters (y_1).

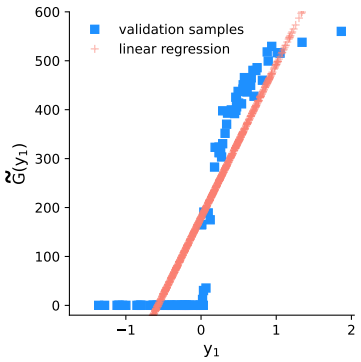
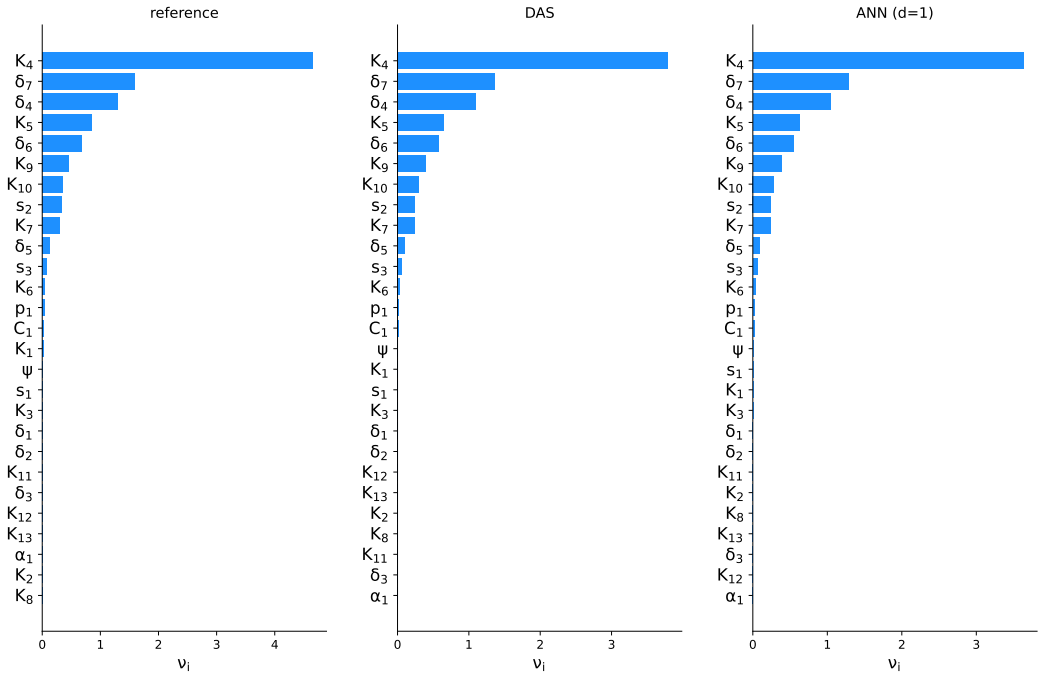


Figure 8. *The linear regression prediction at* $t = 3400$ *days, and* 100 *validation samples* $f(\mathbf{x}_i^*)$, $i = 1, \ldots, 100$, *from* (8.1), *plotted along* $y_1 = \mathbf{w}^T\mathbf{x}^*$, *where here* $\mathbf{w} = \mathbf{a}'/\|\mathbf{a}'\|_2$; *see section* 7.



(a) The reference results. (b) The DAS results. (c) The constrained ANN results

Figure 9. *Global derivative-based sensitivity indices* (6.1) *at* $t = 3400$ *days.*

Finally, we consider a case with a vector-valued QoI, namely, the T-cell count at $t = [5, 15, 24, 38, 40, 45, 50, 55, 50, 65, 90, 140, 500, 750, 1000, 1600, 1800, 2000, 2200, 2400, 2800, 3400]$ days. We also compute the original active subspace where $\overline{C}$ is constructed with finite-difference gradients of $\|\mathbf{f}\|_2$, and we compare this with $\overline{C}_{ANN}$ given by (5.1). Figure 10 shows the corresponding eigenvalues. The decay in this case is less pronounced than for the scalar f, which can be an indication that the active subspace is not the same at every output value. However, since we can still observe a gap after λ_2, we set $d = 2$. As before, we first train a

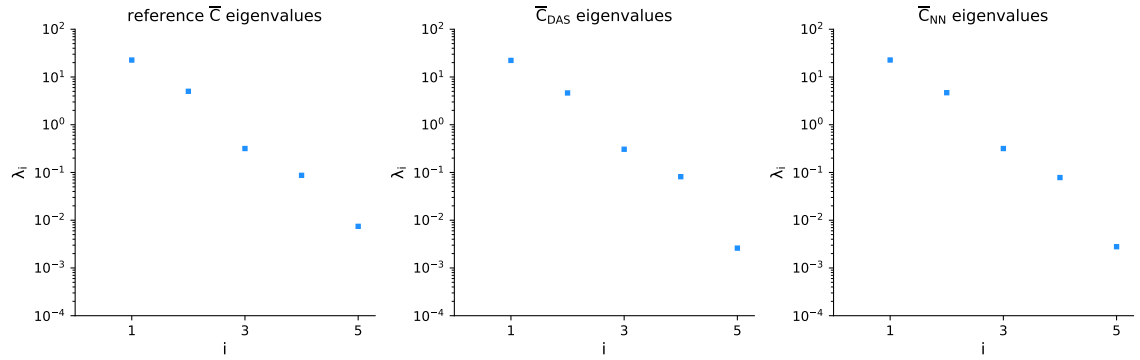


Figure 10. *Left: the first* 5 *reference eigenvalues of* $\overline{C}$*, constructed with finite difference approximations of* $\partial\|\mathbf{f}\|_2/\partial x_i$*. Middle and right: the nonzero eigenvalues of* $\overline{C}_{DAS}$ *and* $\overline{C}_{ANN}$ (5.1) *with* $d = 5$*.*

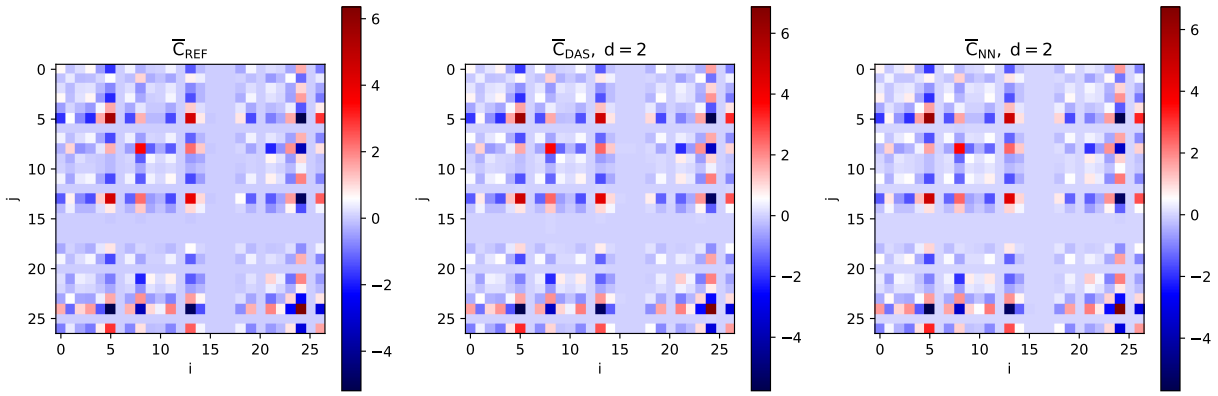
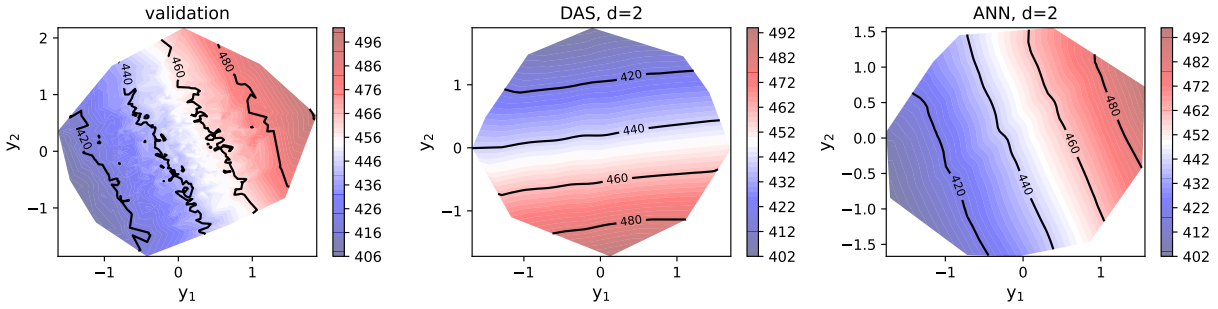


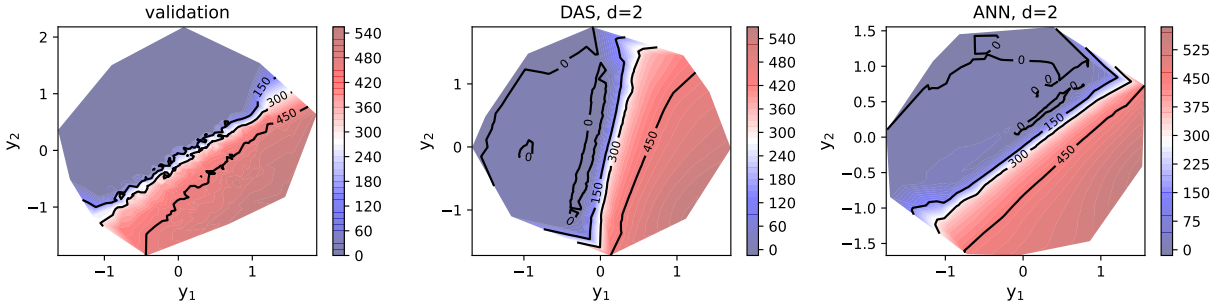
Figure 11. *Heat map of* $\overline{C} = \overline{(\partial\|\mathbf{f}\|_2/\partial\mathbf{x})(\partial\|\mathbf{f}\|_2/\partial\mathbf{x})^T}$ *for the reference active-space method, and the corresponding constrained ANN and DAS approximations based on* $\partial\|\widetilde{\mathbf{f}}\|_2/\partial\mathbf{x}$*. The outputs* $\mathbf{f}$ *and* $\widetilde{\mathbf{f}}$ *were standardized.*

surrogate using $d = 5$ to examine the eigenvalue decay of $\overline{C}_{ANN}$. The 5 nonzero eigenvalues of $\overline{C}_{ANN}$ show similar behavior compared to the reference case, such that we also would have used $d = 2$ without access to the reference.

In Figure 11 we contrast the $\overline{C}$ matrix from the (original) active subspace of $\|\mathbf{f}\|_2$ with its constrained ANN and DAS counterparts, which are a reasonable approximation. Finally, Figure 12 shows the contours of the 2D DAS and constrained ANN surrogates, alongside those obtained from 1000 random validation samples from the code (8.1). We show the results at $t = 45$ and $t = 3400$ days, where both time stamps are outputs from the same vector-valued surrogate. The validation data are plotted in the reference $[y_1, y_2]^T = U_1^T\mathbf{x}$ coordinate system. From the contour lines we can see that the validation data are noisy (as expected), whereas the ($\mathbf{z}$-invariant) surrogates are smooth. Note that at $t = 3400$ days, the surrogates struggle to predict the region where the T-cell count is zero, in the sense that small negative values are present. Since V_1 is similar to U_1 (in the sense both are the dominant eigenvectors from approximately the same matrix), the arbitrary rotation of the constrained ANN $\mathbf{y}$ coordinate system with respect to the reference $\mathbf{y}$ system is usually smaller than that of the DAS system.



(a) $t = 45$ days



(b) $t = 3400$ days

Figure 12. *The contours of the DAS and constrained ANN prediction $\widetilde{G}$ for $d = 2$, and the contours computed from* 1000 *(nonstandardized) validation samples $f(\mathbf{x}_i^*)$, $i = 1, \ldots, 1000$, from* (8.1)*, plotted along $y_1 = U_1^T \mathbf{x}_i^*$. Predictions are shown at two different times, which are outputs from the vector-valued surrogates.*

This is clear from Figure 11. Still, the overall structure of both surrogate responses is very similar to the validation data, and the rotation of the $\mathbf{y}$ system is not of particular interest here.

8.2. CovidSim. CovidSim is an influential individual-based simulation code developed by the MRC centre for global infectious disease analysis at Imperial College London. It has been used to explore various nonpharmaceutical interventions with the aim of reducing the transmission of the coronavirus [18]. Unlike the HIV model, CovidSim requires high-performance computing (HPC) resources if an ensemble of model evaluations is needed. In a previous study, we applied dimension-adaptive SC to this model to estimate the parametric uncertainty due to 19 imperfectly known inputs [16]. This method iteratively refines the sampling plan in important directions, and in [16] we required more than 100 iterations. This may become cumbersome if one is subject to a long queue time before each (relatively small) ensemble is executed on the HPC resource. Active-subspace methods are in this sense more "HPC-friendly", since a single large ensemble can be executed, and the dimension reduction is done in a postprocessing step.

We will consider a case with $D = 51$ continuous input parameters, for which we prescribe uniform input distributions. For a table with all input parameters, their default values, and uncertain range, we refer to the supplementary materials (supplement.pdf [local/web 184KB]).

We sampled the input space with 3000 MC evaluations, and we executed the ensemble on the Altair supercomputer, located at the Poznan Supercomputing and Networking Center. Various components from the VECMA tookit [21] were used to generate the results. In particular, the MC sampling plan was created using EasyVVUQ [33], the workflow and data transfer to and from Altair were handled by FabSim3 [22], and the QCG-PilotJob framework [3] was used to package the 3000 jobs into a single job allocation, thereby circumventing the limit on the maximum number of concurrent jobs that is present on many supercomputers. Each CovidSim sample used 28 cores per node and completed in about 10 minutes, leading to a rough estimate of the computational cost of 14.000 core hours. Using 50 nodes in parallel, the sampling completed in less than a full day. The surrogate is then trained on the EasyVVUQ data frame, using the surrogate modeling component of VECMA tookit [13].

We do not have a reference active-subspace solution for CovidSim, and for conciseness we only show the results of a DAS surrogate. The eigenvalues of the DAS surrogate do show a small gap after λ_1 (see Figure 13, where $d = 5$). That said, since $d = 1$ yielded relatively high training and test errors, we selected $d = 2$ in the end (note that there is also a smaller gap between λ_2 and λ_3). The QoI we used here is the predicted cumulative death count after a simulated time of 801 days. Figure 14 shows the contours in the $\mathbf{y} = W_1^T\mathbf{x}$ coordinate system, for both the DAS surrogate and those obtained from 1500 validation samples. These results again show that the DAS surrogate is a smooth approximation of the code output in the active subspace. To estimate the magnitude of the error, we repeat the replica-based error analysis for the DAS, constrained and unconstrained ANN surrogates (using 100 replica networks with 10 neurons per hidden layer); see Figure 15. Overall, the behavior is similar to that of the HIV case (Figure 6), albeit with a higher error magnitude. The unconstrained ANN error shows a jump from the training to the test set, whereas the DAS and constrained ANN errors are similar, have small confidence intervals and remain almost constant. The similarity aspect again indicates that embedded orthonormality does not influence the training or test performance.

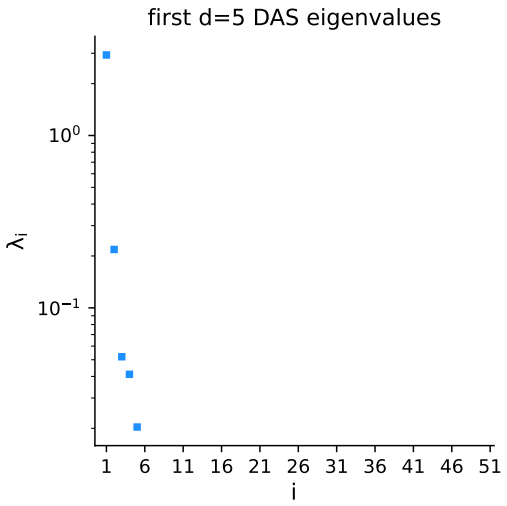


Figure 13. *The nonzero eigenvalues of* $\overline{C}_{ANN}$ (4.5) *for the CovidSim application with* $d = 5$.x.

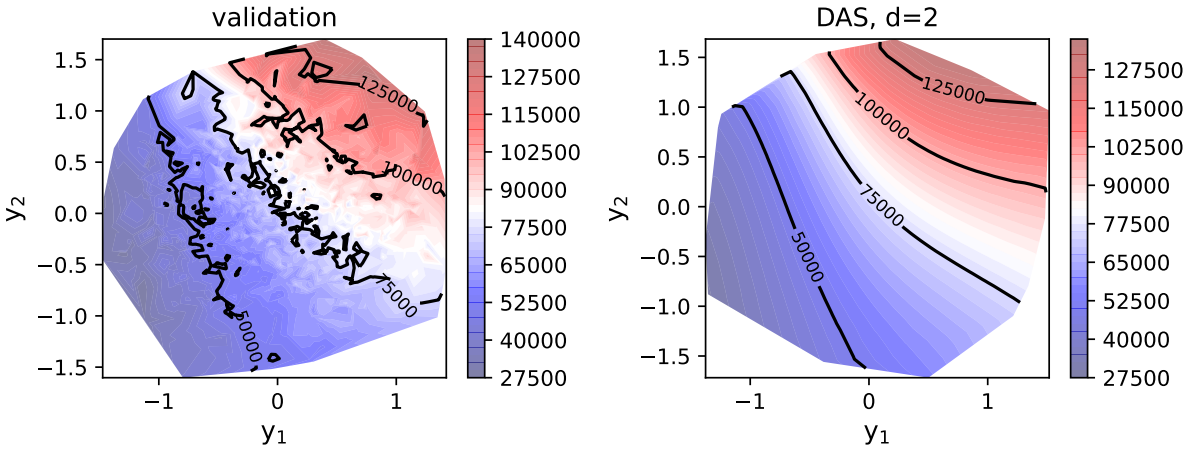


Figure 14. *Right: the contours of the DAS prediction $\widetilde{G}$, trained on 50% of the CovidSim samples with $d = 2$. Left: the contours computed from the remaining 50% (nonstandardized) CovidSim validation samples $f(\mathbf{x}_i^*)$, $i = 1, \ldots, 1500$, plotted along $y_1 = W_1^T \mathbf{x}_i^*$, where $W_1 = [\mathbf{w}_1/\|\mathbf{w}_1\|_2 \ \mathbf{w}_2/\|\mathbf{w}_2\|_2]$ is the orthonormal weight matrix from the DAS surrogate. Predictions are the final cumulative death count after 801 days.*

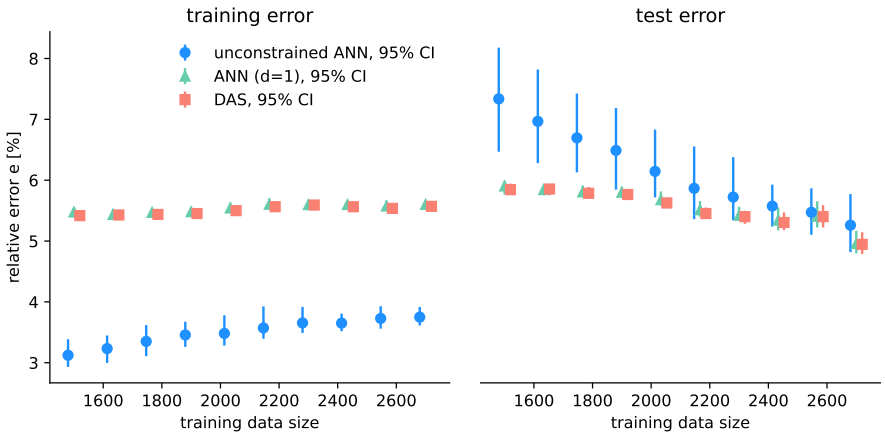


Figure 15. *The training (left) and test (right) 95% error confidence intervals for the DAS, constrained and unconstrained ANN surrogates versus the training data size for the CovidSim application, using 10 neurons for the hidden layers.*

8.3. Code and data availability. We implemented the DAS method in the VECMA toolkit [13]. The Python codes and the data required to reproduce all results presented above can be found at [15]. The Gram–Schmidt derivatives are available separately at [14].

9. Conclusion. We examined the DAS method, which is a derivative-free active-subspace method based on neural networks. The main components of this method are contained in the first hidden layer, where the number of neurons is restricted to the dimension of the active subspace (d), and the weight matrix is kept orthonormal via Gram–Schmidt. This mimics the original active-subspace method (which we consider as the reference solution), where an orthonormal matrix is used to project the high-dimensional input vector to a

low-dimensional active subspace. Several new theoretical investigations were performed. First, the back-propagation step requires the derivative of Gram–Schmidt vectors with respect to the original, nonorthogonal basis vector, for which we have derived a new analytic recurrence relation, which might prove useful outside the DAS method. We further showed that the DAS network is $\mathbf{z}$-invariant, i.e., has at most d nonzero eigenvalues. We also note that unlike the original active-subspace method, the DAS network can easily handle vector-valued QoI. That said, the performance can degrade if vector entries have different active subspaces. Additionally, we investigated a constrained neural network, where the number of neurons in the first hidden layer is still restricted to d, but where orthonormality of the corresponding weight matrix is not enforced during training. We can still extract an orthonormal projection matrix in a postprocessing step.

We first assessed the performance of both the DAS method and the constrained neural network on an HIV model with 27 input parameters, for which we have an (active-subspace) reference solution. Both methods gave a good approximation of the reference, despite the lack of direct gradient data. The DAS method and constrained neural network generated a surrogate model with $d = 1$ which was more robust than a classical (unconstrained) neural network, in the sense that its training error closely matched the test error, at various training/test data splits. Since the DAS and constrained neural network only differ by the enforcement of orthonormality, this is an indication that the good generalization properties are a result of restricting the neurons to the active-subspace dimension in the first hidden layer, and not due to enforced orthonormality during training. By repeating the error analysis 100 times, we also found that the training and test errors from the DAS and constrained neural network showed (very) little variation compared to those from a standard ANN. Our final application was a computationally expensive COVID19 model with 51 inputs, where high-performance computing resources were required to sample the input space. We found similar results compared to the HIV application, except that the model response could only be reasonably approximated in a 2D active subspace.

Appendix A. Gram–Schmidt derivative derivation. We will first compute the derivative of $\mathbf{w}_i$, before it is normalized by its length $\|\mathbf{w}_i\|_2$:

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_k} = \frac{\partial \mathbf{q}_i}{\partial \mathbf{q}_k} - \sum_{j=1}^{i-1} \frac{\partial}{\partial \mathbf{q}_k} \left[\left(\frac{\mathbf{w}_j^T \mathbf{q}_i}{\mathbf{w}_j^T \mathbf{w}_j} \right) \mathbf{w}_j \right], \quad i = 1, \ldots, d, \quad k = 1, \ldots, d. \tag{A.1}$$

A brute-force computation of this derivative using a computer algebra system will show that this quickly becomes a complicated expression with a very large number of terms. However, we can find a simple expression for this derivative by computing it in an iterative fashion:

$$i = 1: \quad \mathbf{w}_1 = \mathbf{q}_1 \Rightarrow \frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_1} = I_D =: D_{11} \quad \text{and} \quad \frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_i} = 0 \quad \text{for } i > 1, \tag{A.2}$$

$$i = 2: \quad \mathbf{w}_2 = \mathbf{q}_2 - \left(\frac{\mathbf{w}_1^T \mathbf{q}_2}{\mathbf{w}_1^T \mathbf{w}_1} \right) \mathbf{w}_1. \tag{A.3}$$

First we compute the shear derivative $\partial \mathbf{w}_2/\partial \mathbf{q}_1$ (defined as $\partial \mathbf{w}_i/\partial \mathbf{q}_k$, where $i \neq k$):

$$\begin{aligned}\frac{\partial \mathbf{w}_2}{\partial \mathbf{q}_1} &= -\frac{\partial}{\partial \mathbf{q}_1}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_2}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1\right] \\ &= -\frac{\partial}{\partial \mathbf{w}_1}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_2}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1\right]\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_1} \\ &= \underbrace{-\left[\frac{1}{\mathbf{w}_1^T\mathbf{w}_1}\,\mathbf{w}_1\mathbf{q}_2^T - \frac{2\mathbf{w}_1^T\mathbf{q}_2}{(\mathbf{w}_1^T\mathbf{w}_1)^2}\,\mathbf{w}_1\mathbf{w}_1^T + \frac{\mathbf{w}_1^T\mathbf{q}_2}{\mathbf{w}_1^T\mathbf{w}_1}\,I_D\right]}_{=:D_{21}}\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}1} \\ &= D_{21}\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_1} = D_{21}D_{11}. \end{aligned} \tag{A.4}$$

Here, the second equality is just the chain rule, and in the third we used the following matrix calculus identity:

$$D_{ij} := -\frac{\partial}{\partial \mathbf{w}_j}\left[\left(\frac{\mathbf{w}_j^T\mathbf{q}_i}{\mathbf{w}_j^T\mathbf{w}_j}\right)\mathbf{w}_j\right] = -\left[\frac{1}{\mathbf{w}_j^T\mathbf{w}_j}\,\mathbf{w}_j\mathbf{q}_i^T - \frac{2\mathbf{w}_j^T\mathbf{q}_i}{(\mathbf{w}_j^T\mathbf{w}_j)^2}\,\mathbf{w}_j\mathbf{w}_j^T + \frac{\mathbf{w}_j^T\mathbf{q}_i}{\mathbf{w}_j^T\mathbf{w}_j}\,I_D\right], \tag{A.5}$$

which is derived in Appendix B. Hence, we can compute D_{21}, and the matrix $D_{11} := \partial \mathbf{w}_1/\partial \mathbf{q}_1 = I_D$ was already computed in the previous iteration. The matrix-matrix product of D_{21} and D_{11} yields $\partial \mathbf{w}_2/\partial \mathbf{q}_1$.

We now compute the normal derivative $\partial \mathbf{w}_2/\partial \mathbf{q}_2$:

$$\begin{aligned}\frac{\partial \mathbf{w}_2}{\partial \mathbf{q}_2} &= I_D - \frac{\partial}{\partial \mathbf{q}_2}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_2}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1\right] \\ &= I_D - \frac{\mathbf{w}_1\mathbf{w}_1^T}{\mathbf{w}_1^T\mathbf{w}_1} \\ &= D_{11} - \frac{\mathbf{w}_1\mathbf{w}_1^T}{\mathbf{w}_1^T\mathbf{w}_1} =: D_{22}. \end{aligned} \tag{A.6}$$

In the second equality we made use of the identity:

$$\frac{\partial}{\partial \mathbf{q}_i}\left[\left(\frac{\mathbf{w}_j^T\mathbf{q}_i}{\mathbf{w}_j^T\mathbf{w}_j}\right)\mathbf{w}_j\right] = \frac{\mathbf{w}_j\mathbf{w}_j^T}{\mathbf{w}_j^T\mathbf{w}_j}, \tag{A.7}$$

also derived in Appendix B. This holds if $\mathbf{w}_j$ does not depend upon $\mathbf{q}_i$, which is true in (A.6) since $\mathbf{w}_1 = \mathbf{w}_1(\mathbf{q}_1)$. Also, since $\mathbf{w}_2 = \mathbf{w}_2(\mathbf{q}_1, \mathbf{q}_2)$, we have $\partial \mathbf{w}_2/\partial \mathbf{q}_k = 0$ for $k > 2$. Further note that in (A.6), we can write $\partial \mathbf{w}_2/\partial \mathbf{q}_2$ as the difference between $D_{11} := \partial \mathbf{w}_1/\partial \mathbf{q}_1$ and $\mathbf{w}_1\mathbf{w}_1^T/\mathbf{w}_1^T\mathbf{w}_1$, and that we have defined this expression for $\partial \mathbf{w}_2/\partial \mathbf{q}_2$ as D_{22}.

$i = 3$:

$$\mathbf{w}_3 = \mathbf{q}_3 - \left(\frac{\mathbf{w}_1^T\mathbf{q}_3}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1 - \left(\frac{\mathbf{w}_2^T\mathbf{q}_3}{\mathbf{w}_2^T\mathbf{w}_2}\right)\mathbf{w}_2. \tag{A.8}$$

We again compute the shear derivatives first:

$$\begin{aligned}\frac{\partial \mathbf{w}_3}{\partial \mathbf{q}_1} &= -\frac{\partial}{\partial \mathbf{q}_1}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_3}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1\right] - \frac{\partial}{\partial \mathbf{q}_1}\left[\left(\frac{\mathbf{w}_2^T\mathbf{q}_3}{\mathbf{w}_2^T\mathbf{w}_2}\right)\mathbf{w}_2\right] \\ &= \underbrace{-\frac{\partial}{\partial \mathbf{w}_1}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_3}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1\right]}_{=:D_{31}}\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_1} \underbrace{-\frac{\partial}{\partial \mathbf{w}_2}\left[\left(\frac{\mathbf{w}_2^T\mathbf{q}_3}{\mathbf{w}_2^T\mathbf{w}_2}\right)\mathbf{w}_2\right]}_{=:D_{32}}\frac{\partial \mathbf{w}_2}{\partial \mathbf{q}_1} \\ &= D_{31}\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_1} + D_{32}\frac{\partial \mathbf{w}_2}{\partial \mathbf{q}_1} = D_{31}D_{11} + D_{32}D_{21}D_{11}; \end{aligned} \tag{A.9}$$

and

$$\begin{aligned}\frac{\partial \mathbf{w}_3}{\partial \mathbf{q}_2} &= -\frac{\partial}{\partial \mathbf{q}_2}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_3}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1\right] - \frac{\partial}{\partial \mathbf{q}_2}\left[\left(\frac{\mathbf{w}_2^T\mathbf{q}_3}{\mathbf{w}_2^T\mathbf{w}_2}\right)\mathbf{w}_2\right] \\ &= -\frac{\partial}{\partial \mathbf{w}_1}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_3}{\mathbf{w}_1^T\mathbf{w}_1}\right)\mathbf{w}_1\right]\underbrace{\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_2}}_{0} \underbrace{-\frac{\partial}{\partial \mathbf{w}_2}\left[\left(\frac{\mathbf{w}_2^T\mathbf{q}_3}{\mathbf{w}_2^T\mathbf{w}_2}\right)\mathbf{w}_2\right]}_{:=D_{32}}\frac{\partial \mathbf{w}_2}{\partial \mathbf{q}_2} \\ &= D_{32}\frac{\partial \mathbf{w}_2}{\partial \mathbf{q}_2} = D_{32}D_{22}. \end{aligned} \tag{A.10}$$

The normal derivative is given by

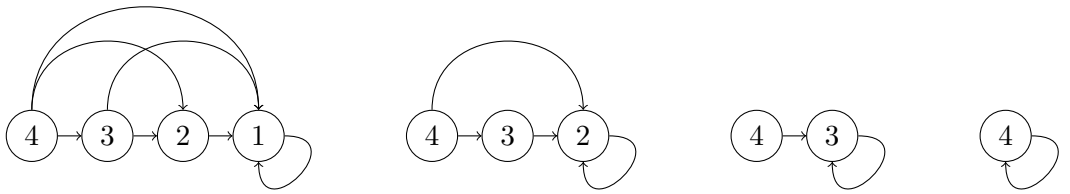
$$\begin{aligned}\frac{\partial \mathbf{w}_3}{\partial \mathbf{q}_3} &= I_D - \frac{\partial}{\partial \mathbf{q}_3}\left[\left(\frac{\mathbf{w}_1^T\mathbf{q}_3}{\mathbf{w}_1^T\mathbf{w}_1}\right)\right] - \frac{\partial}{\partial \mathbf{q}_3}\left[\left(\frac{\mathbf{w}_2^T\mathbf{q}_3}{\mathbf{w}_2^T\mathbf{w}_2}\right)\right] \\ &= \underbrace{I_D - \frac{\mathbf{w}_1\mathbf{w}_1^T}{\mathbf{w}_1^T\mathbf{w}_1}}_{D_{22}} - \frac{\mathbf{w}_2\mathbf{w}_2^T}{\mathbf{w}_2^T\mathbf{w}_2} = D_{22} - \frac{\mathbf{w}_2\mathbf{w}_2^T}{\mathbf{w}_2^T\mathbf{w}_2} =: D_{33}. \end{aligned} \tag{A.11}$$

Finally, consider the case for $i = 4$ in shorthand notation only:

$$\begin{aligned}\frac{\partial \mathbf{w}_4}{\partial \mathbf{q}_1} &= D_{41}D_{11} + D_{42}D_{21}D_{11} + D_{43}D_{31}D_{11} + D_{43}D_{32}D_{21}D_{11}, \\ \frac{\partial \mathbf{w}_4}{\partial \mathbf{q}_2} &= D_{42}D_{22} + D_{43}D_{32}D_{22}, \\ \frac{\partial \mathbf{w}_4}{\partial \mathbf{q}_3} &= D_{43}D_{33}, \\ \frac{\partial \mathbf{w}_4}{\partial \mathbf{q}_3} &= D_{33} - \frac{\mathbf{w}_3\mathbf{w}_3^T}{\mathbf{w}_3^T\mathbf{w}_3} =: D_{44}. \end{aligned} \tag{A.12}$$

The structure is now apparent. The gradients $\partial \mathbf{w}_i/\partial \mathbf{q}_k$, when completely expanded as in (A.12), are determined by a series of matrix-matrix multiplications, the indices of which come from all pairwise paths of a directed graph from $i \to k$, ending with $k \to k$. To see this, consider the graphs of Figure 16, and compare this to the indices of the D_{ij} matrices appearing in the expressions of (A.12).

The graph gives some insight into the structure of each derivative, as it allows us to directly expand all terms that make up each gradient. However, we will not directly use it



(a) All pairwise paths from 4 to 1, always ending with $(1,1)$. Corresponds to indices of $\partial \mathbf{w}_4/\partial \mathbf{q}_1$.

(b) All pairwise paths from 4 to 2, always ending with $(2,2)$. Corresponds to indices of $\partial \mathbf{w}_4/\partial \mathbf{q}_2$.

(c) All pairwise paths from 4 to 3, always ending with $(3,3)$. Corresponds to indices of $\partial \mathbf{w}_4/\partial \mathbf{q}_3$.

(d) The only pairwise paths from 4 to 4. Corresponds to index of $\partial \mathbf{w}_4/\partial \mathbf{q}_4$.

Figure 16. *The directed graphs that generate the indices of the D_{ij} matrices of* (A.12).

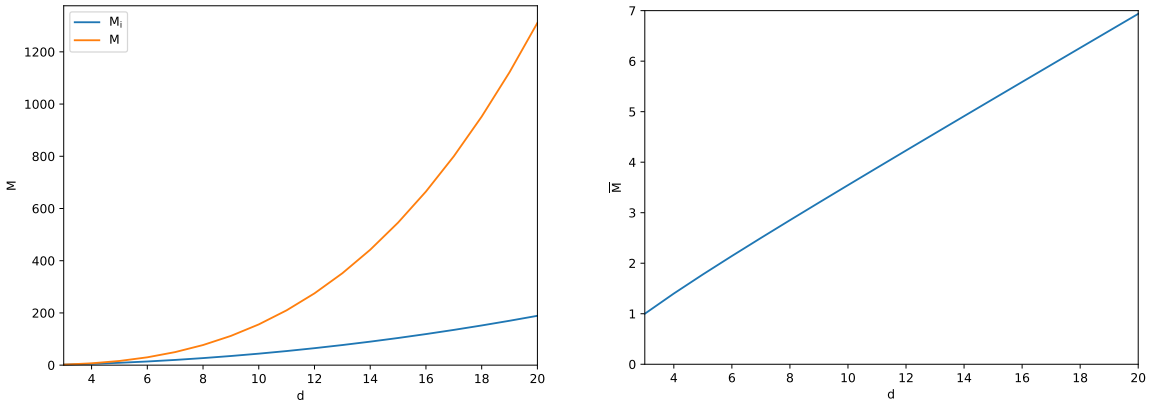
in practice to compute the gradients. Instead, we will start with the gradient of $\mathbf{w}_1$, then compute the gradients of $\mathbf{w}_2$, etc. This is because at any given $\mathbf{w}_i$, we can reuse the results from the previous iteration, thereby avoiding repeated matrix multiplications. This is clear when we write the shear gradients as

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_k} = \sum_{j=1}^{i-1} D_{ij} \frac{\partial \mathbf{w}_j}{\partial \mathbf{q}_k}, \quad i \neq k, \;\; i > k. \tag{A.13}$$

At any given $i > 1$, all $\partial \mathbf{w}_j/\partial \mathbf{q}_k$ terms above have either already been computed at the previous iterations, or are zero when $k > j$. If we count the minimum number of matrix-matrix multiplications that are required to compute all shear gradients at a given i, we find that when $k = 1$, we get $i - 1$ matrix-matrix multiplications. As an example, consider $\partial \mathbf{w}_4/\partial \mathbf{q}_1 = D_{41}\partial \mathbf{w}_1/\partial \mathbf{q}_1 + D_{42}\partial \mathbf{w}_2/\partial \mathbf{q}_1 + D_{43}\partial \mathbf{w}_3/\partial \mathbf{q}_1$, which requires $i - 1 = 3$ matrix multiplications. Technically however, D_{41} is multiplied by $\partial \mathbf{w}_1/\partial \mathbf{q}_1 = I_D$, so the first product we never have to compute. When $k = 2$, we still have $i-1$ terms in (A.13). However, the first term will include $\partial \mathbf{w}_1/\partial \mathbf{q}_2 = 0$. Likewise, when $k = 3$ the first two terms will be zero. Thus the total number of required matrix multiplications M_i is $M_i = (i-1)+(i-2)+\cdots+2+1-1$, where we subtracted 1 at the end because we do not count multiplication by I_D. Finally, the total number of matrix multiplications M to compute all nonzero gradients of $\mathbf{w}_i$, for all $i = 1, \ldots, d$, is therefore given by

$$M = \sum_{i=3}^{d} M_i = \sum_{i=3}^{d} \sum_{j=2}^{i-1} j. \tag{A.14}$$

We start counting at $i = 3$ because $i = 1$ and $i = 2$ require no matrix-matrix multiplications; see (A.2) and (A.4). In Figure 17(a) we plot M versus d which shows that for $d = 20$ we would already need to perform more than 1200 matrix-matrix multiplications. That said, it should be noted that this is the cost of computing *all* nonzero gradients $\partial \mathbf{w}_i/\partial \mathbf{q}_k$ which require matrix-matrix multiplication. In the case of $d = 20$, the total number of such gradients is



(a) The total number of matrix multiplication M_i at a given i, and the total cumulative cost M.

(b) The total number of matrix-matrix multiplications, divided by the number of gradients that are computed.

Figure 17. *The number of matrix-matrix multiplications as a function of d.*

given by $2+3+\cdots+19 = 189$. Hence, it is not particularly surprising that a large number of multiplications are required. Figure 17(b) shows the total number of multiplications divided by the number of gradients which are computed, which suggests that the average number of matrix-matrix multiplications scales linearly with d. Still, for high d the number of matrix-matrix multiplications M can be significant.

The preceding pertained to the shear gradients. From (A.4), (A.11), and (A.12), we can see that the normal gradients are computed as

$$D_{ii} := \frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_i} = D_{i-1,i-1} - \frac{\mathbf{w}_{i-1}\mathbf{w}_{i-1}^T}{\mathbf{w}_{i-1}^T\mathbf{w}_{i-1}}, \quad i > 1, \tag{A.15}$$

with $D_{11} := I_D$. Note that no matrix-matrix multiplication is involved in computing these derivatives.

Finally, let us note that we verified the correctness of (A.13) and (A.15) by comparing our numerical result with the results from a computer algebra system, which used symbolic math to directly compute the derivatives from the definition of the $\mathbf{w}_i$ (3.1).

Appendix B. Matrix calculus identities. We will derive a number of useful matrix calculus identities here. Let $\mathbf{w}$ and $\mathbf{q}$ be two vectors in $\mathbb{R}^D$. Then

$$\frac{\partial}{\partial \mathbf{w}}\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right) = \frac{1}{\mathbf{w}^T\mathbf{w}}\,\mathbf{q}^T - \frac{2\,\mathbf{w}^T\mathbf{q}}{(\mathbf{w}^T\mathbf{w})^2}\,\mathbf{w}^T \in \mathbb{R}^{1\times D}. \tag{B.1}$$

Proof. This follows directly from the quotient rule:

$$\frac{\partial}{\partial \mathbf{w}}\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right) = \frac{\frac{\partial}{\partial \mathbf{w}}[\mathbf{w}^T\mathbf{q}]\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\mathbf{q}\frac{\partial}{\partial \mathbf{w}}[\mathbf{w}^T\mathbf{w}]}{(\mathbf{w}^T\mathbf{w})^2} = \frac{\mathbf{q}^T\,(\mathbf{w}^T\mathbf{w}) - (\mathbf{w}^T\mathbf{q})\,2\mathbf{w}^T}{(\mathbf{w}^T\mathbf{w})^2} \tag{B.2}$$

which yields (B.1). Going one step further, we have the following identity

$$\frac{\partial}{\partial \mathbf{w}}\left[\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right)\mathbf{w}\right] = \frac{1}{\mathbf{w}^T\mathbf{w}}\,\mathbf{w}\mathbf{q}^T - \frac{2\mathbf{w}^T\mathbf{q}}{(\mathbf{w}^T\mathbf{w})^2}\,\mathbf{w}\mathbf{w}^T + \frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\,I_D \in \mathbb{R}^{D\times D}. \tag{B.3}$$

Proof. Apply the product rule:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}}\left[\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right)\mathbf{w}\right] &= \mathbf{w}\frac{\partial}{\partial \mathbf{w}}\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right) + \left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right)\,I_D \\ &= \mathbf{w}\left[\frac{1}{\mathbf{w}^T\mathbf{w}}\,\mathbf{q}^T - \frac{2\,\mathbf{w}^T\mathbf{q}}{(\mathbf{w}^T\mathbf{w})^2}\,\mathbf{w}^T\right] + \left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right)\,I_D \\ &= \frac{1}{\mathbf{w}^T\mathbf{w}}\,\mathbf{w}\mathbf{q}^T - \frac{2\mathbf{w}^T\mathbf{q}}{(\mathbf{w}^T\mathbf{w})^2}\,\mathbf{w}\mathbf{w}^T + \frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\,I_D, \end{aligned} \tag{B.4}$$

where in the second equality we inserted (B.1). One more identity we require is given by

$$\frac{\partial}{\partial \mathbf{q}}\left[\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right)\mathbf{w}\right] = \frac{1}{\mathbf{w}^T\mathbf{w}}\,\mathbf{w}\mathbf{w}^T \in \mathbb{R}^{D\times D}. \tag{B.5}$$

This holds if $\mathbf{w}$ does not depend upon $\mathbf{q}$.

Proof. Apply the product rule:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{q}}\left[\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right)\mathbf{w}\right] &= \mathbf{w}\frac{\partial}{\partial \mathbf{q}}\left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right) + \left(\frac{\mathbf{w}^T\mathbf{q}}{\mathbf{w}^T\mathbf{w}}\right)\underbrace{\frac{\partial \mathbf{w}}{\partial \mathbf{q}}}_{0} \\ &= \frac{1}{\mathbf{w}^T\mathbf{w}}\,\mathbf{w}\frac{\partial}{\partial \mathbf{q}}\left(\mathbf{w}^T\mathbf{q}\right) = \frac{1}{\mathbf{w}^T\mathbf{w}}\,\mathbf{w}\mathbf{w}^T. \end{aligned} \tag{B.6}$$

Since we assume that $\mathbf{w}$ does not depend upon $\mathbf{q}$, we could take $1/\mathbf{w}^T\mathbf{w}$ out of the differentiation operator. This is perhaps confusing, as we do not make this assumption elsewhere. However, in the context of deriving the derivatives of the Gram–Schmidt vectors, this assumption always holds when we have to apply (B.5); see, e.g., (A.6).

Finally, a standard formula (see, e.g., [31]), for the gradient of a normed vector is

$$\frac{\partial}{\partial \mathbf{w}}\left(\frac{\mathbf{w}}{\|\mathbf{w}\|_2}\right) = \frac{I_D}{\|\mathbf{w}\|_2} - \frac{\mathbf{w}\mathbf{w}^T}{\|\mathbf{w}\|_2^3}. \tag{B.7}$$

In our case, a useful related identity is

$$\frac{\partial}{\partial \mathbf{q}}\left(\frac{\mathbf{w}}{\|\mathbf{w}\|_2}\right) = \left[\frac{I_D}{\|\mathbf{w}\|_2} - \frac{\mathbf{w}\mathbf{w}^T}{\|\mathbf{w}\|_2^3}\right]\frac{\partial \mathbf{w}}{\partial \mathbf{q}}, \tag{B.8}$$

which follows directly from the chain rule.

Acknowledgments. The CovidSim calculations were performed at the Poznan Supercomputing and Networking Center. I thank the EPSRC for funding the Software Environment for Actionable and VVUQ-evaluated Exascale Applications (SEAVEA) grant (EP/W007711/1). Finally, I would like to thank three anonymous reviewers for their insightful comments.

REFERENCES

[1] C. C. Aggarwal, *Neural Networks and Deep Learning*, Springer Nature Switzerland, 2018, Ch. 3.
[2] G. Blatman and B. Sudret, *Adaptive sparse polynomial chaos expansion based on least angle regression*, J. Comput. Phys., 230 (2011), pp. 2345–2367.
[3] B. Bosak, T. Piontek, P. Karlshoefer, E. Raffin, J. Lakhlili, and P. Kopta, *Verification, validation and uncertainty quantification of large-scale applications with QCG-PilotJob*, in International Conference on Computational Science, Springer, Cham, Switzerland, 2021, pp. 495–501.
[4] P. Constantine, *A Quick-and-Dirty Check for a One-Dimensional Active Subspace*, preprint, arXiv:1402.3838, 2014.
[5] P. Constantine, *Active Subspace Datasets*, https://github.com/paulcon/as-data-sets (2021).
[6] P. Constantine and P. Diaz, *Global sensitivity metrics from active subspaces*, Reliab. Eng. Syst. Safety, 162 (2017), pp. 1–13.
[7] P. Constantine, A. Eftekhari, and M. Wakin, *Computing active subspaces efficiently with gradient sketching*, in 2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), IEEE, Piscataway, NJ, 2015, pp. 353–356.
[8] P. Constantine, M. Emory, J. Larsson, and G. Iaccarino, *Exploiting active subspaces to quantify uncertainty in the numerical simulation of the HyShot II scramjet*, J. Comput. Phys., 302 (2015), pp. 1–20.
[9] P. G. Constantine, E. Dow, and Q. Wang, *Active subspace methods in theory and practice: Applications to kriging surfaces*, SIAM J. Sci. Comput., 36 (2014), pp. A1500–A1524.
[10] C. Cui, K. Zhang, T. Daulbaev, J. Gusak, I. Oseledets, and Z. Zhang, *Active subspace of neural networks: Structural analysis and universal attacks*, SIAM J. Math. Data Sci., 2 (2020), pp. 1096–1122.
[11] Z. del Rosario, P. Constantine, and G. Iaccarino, *Developing design insight through active subspaces*, in 19th AIAA Non-Deterministic Approaches Conference, AIAA 2017-1090, AIAA, Reston, VA, 2017.
[12] Y. Dimopoulos, P. Bourret, and S. Lek, *Use of some sensitivity criteria for choosing networks with good generalization ability*, Neural Process. Lett., 2 (1995), pp. 1–4.
[13] W. Edeling, *EasySurrogate*, https://github.com/wedeling/EasySurrogate (2021).
[14] W. Edeling, *wedeling/Gram_Schmidt_Derivatives: Release with DAS Article*, https://doi.org/10.5281/zenodo.5734109 (2021).
[15] W. Edeling, *wedeling/deep_active_subspace_data: Final Revision of DAS Article*, https://doi.org/10.5281/zenodo.7009160 (2022).
[16] W. Edeling, H. Arabnejad, R. Sinclair, D. Suleimenova, K. Gopalakrishnan, B. Bosak, D. Groen, I. Mahmood, D. Crommelin, P. V. Coveney, *The impact of uncertainty on predictions of the CovidSim epidemiological code*, Nature Comput. Sci., 1 (2021), pp. 128–135.
[17] M. Eldred and J. Burkardt, *Comparison of non-intrusive polynomial chaos and stochastic collocation methods for uncertainty quantification*, in 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, AIAA 2009-976, AIAA, Reston, VA, 2009.
[18] N. Ferguson et al., *Report 9: Impact of Non-pharmaceutical Interventions (NPIs) to Reduce COVID19 Mortality and Healthcare Demand*, https://doi.org/10.25561/77482 (2020).
[19] T. Gerstner and M. Griebel, *Dimension-adaptive tensor-product quadrature*, Computing, 71 (2003), pp. 65–87.
[20] Z. Grey and P. Constantine, *Active subspaces of airfoil shape parameterizations*, AIAA J., 56 (2018), pp. 2003–2017.

[21] D. Groen et al., *VECMAtk: A scalable verification, validation and uncertainty quantification toolkit for scientific simulations*, Philos. Trans. Roy. Soc. A, 379 (2021), 20200221.
[22] D. Groen, A. Bhati, J. Suter, J. Hetherington, S. Zasada, and P. Coveney, *FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures*, Comput. Phys. Commun., 207 (2016), pp. 375–385.
[23] H. Guy, A. Alexanderian, and M. Yu, *A distributed active subspace method for scalable surrogate modeling of function valued outputs*, J. Sci. Comput., 85 (2020), pp. 1–25.
[24] W. He, Y. Zeng, and G. Li, *An adaptive polynomial chaos expansion for high-dimensional reliability analysis*, Struct. Multidiscip. Optim., 62 (2020), pp. 2051–2067.
[25] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadephia, 2002, Ch. 19.
[26] J. Jakeman, M. Eldred, G. Geraci, and A. Gorodetsky, *Adaptive multi-index collocation for uncertainty quantification and sensitivity analysis*, Internat. J. Numer. Methods Engrg., 121 (2020), pp. 1314–1343.
[27] J. Jefferson, J. Gilbert, P. Constantine, and R. Maxwell, *Active subspaces for sensitivity analysis and dimension reduction of an integrated hydrologic model*, Comput. Geosci., 83 (2015), pp. 127–138.
[28] X. Liu and S. Guillas, *Dimension reduction for Gaussian process emulation: An application to the influence of bathymetry on tsunami heights*, SIAM/ASA J. Uncertainty Quantif., 5 (2017), pp. 787–812.
[29] T. Loudon and S. Pankavich, *Mathematical analysis and dynamic active subspaces for a long term model of HIV*, Math. Biosci. Eng., 14 (2017), pp. 709–733.
[30] X. Ma and N. Zabaras, *An adaptive high-dimensional stochastic model representation technique for the solution of stochastic partial differential equations*, J. Comput. Phys., 229 (2010), pp. 3884–3915.
[31] K. Petersen and M. Pedersen, *The Matrix Cookbook,* version 20121115, Technical report 3274, Technical University Denmark, Kongens Lyngby, Denmark, 2012.
[32] H. Rabitz and Ö. Aliş, *General foundations of high-dimensional model representations*, J. Math. Chem., 25 (1999), pp. 197–233.
[33] R. Richardson, D. Wright, W. Edeling, V. Jancauskas, J. Lakhlili, and P. Coveney, *EasyVVUQ: A library for verification, validation and uncertainty quantification in high performance computing*, J. Open Res. Softw., 8 (2020), 11, https://doi.org/10.5334/jors.303.
[34] I. Sobol, *Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates*, Math. Comput. Simulation, 55 (2001), pp. 271–280.
[35] I. Sobol and S. Kucherenko, *Derivative based global sensitivity measures and their link with global sensitivity indices*, Math. Comput. Simulation, 79 (2009), pp. 3009–3017, https://doi.org/10.1016/j.matcom.2009.01.023.
[36] E. Stevens, L. Antiga, and T. Viehmann, *Deep Learning with PyTorch*, Manning Publications, Shelter Island, NY, 2020.
[37] R. Tripathy and I. Bilionis, *Deep active subspaces: A scalable method for high-dimensional uncertainty propagation*, in ASME, 2019 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers Digital Collection, ASME, New York, 2019.
[38] R. Tripathy, I. Bilionis, and M. Gonzalez, *Gaussian processes with built-in dimensionality reduction: Applications to high-dimensional uncertainty propagation*, J. Comput. Phys., 321 (2016), pp. 191–223.
[39] O. Zahm, P. G. Constantine, C. Prieur, and Y. M. Marzouk, *Gradient-based dimension reduction of multivariate vector-valued functions*, SIAM J. Sci. Comput., 42 (2020), pp. A534–A558.