Supplementary Information for "Neural Decomposition: Functional ANOVA with Variational Autoencoders"

A Experimental details

A.1 Synthetic data generative mechanism

We used the following data generative mechanism. For $i = 1, \ldots, N$ ($N = 500$) we generated

- $z_i \sim U(-2, 2)$
- $c_i \sim U(-2, 2)$
- $y_i^{(1)} := \exp(-z_i^2) + 0.3\tanh(x) + \varepsilon_i$
- $y_i^{(2)} := \sin(z_i) + 0.2x_i + 0.2\sin(z_i) \cdot x_i \cdot I(z_i > 0) + \varepsilon_i$

A.2 Synthetic data generative mechanism

- $z_i \sim U(-2, 2)$ for $i = 1, ..., N$
- $c_i \sim U(-2, 2)$ for $i = 1, ..., N$
- for features $j = 1, \ldots 5$
  - $y^{(j)} := w_j \cdot \cos(z) + \varepsilon$ where $w_j \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$
- for features $j = 6, \ldots 10$
  - $y^{(j)} := 0.5z + w_j c + \varepsilon$ where $w_j \in \{0.05, 0.1, 0.15, 0.2, 0.25\}$
- for features $j = 11, \ldots 15$
  - $y^{(j)} := w_j \tanh(z)c + \varepsilon$ where $w_j \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$
- for features $j = 16, \ldots 20$
  - $y^{(j)} := w_j c + 0.01(0.12 - w_j)z\tanh(c) + \varepsilon$ where $w_j \in \{0.02, 0.04, 0.06, 0.08, 0.1\}$
- for features $j = 21, \ldots 25$
  - $y_i^{(j)} := 0.1\tanh(z) + 0.2\tanh(c) + w_j \sin(z)x + \varepsilon$ where $w_j \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$

  with noise $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.05$

A.3 Synthetic data generative mechanism for the batch effect (a two-dimensional latent space)

To simulate a batch effect (here for two known batches), we generated a latent (z_1, z_2) space together with batch indicators c for $i = 1, ..., N$ $(N = 1000)$

- $z_{1,i} \sim U(-2, 2)$ for $i = 1, ..., N$
- $z_{2,i} \sim U(-2, 2)$ for $i = 1, ..., N$
- $c_i \sim \text{Bernoulli}(0.5)$ for $i = 1, ..., N$
- for features $j = 1, \dots 5$
  - $y^{(j)} := 0.3w_j \tanh(z_1) + 0.2w_j \exp(-0.5z_2^2) + 0.3w_j c + \varepsilon$ where $w_j \in \{1, 2, 3, 4, 5\}$
- for features $j = 6, \dots 10$
  - $y^{(j)} := 0.2w_j \tanh(z_2) + 0.4(6 - w_j)c + \varepsilon$ where $w_j \in \{1, 2, 3, 4, 5\}$
- for features $j = 11, \dots 15$
  - $y^{(j)} := w_j z_1 + (0.6 - w_j)z_2 + (0.6 - w_j)\tanh(z_1)c + \varepsilon$ where $w_j \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$
- for features $j = 16, \dots 20$
  - $y^{(j)} := \tanh(2z_1) + w_j \tanh(z_2) + \varepsilon$ where $w_j \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$
- for features $j = 21, \dots 25$
  - $y_i^{(j)} := 0.1c + w_j \tanh(z_1)c + \varepsilon$ where $w_j \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$

  with noise $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.05$

B Additional results on single-cell data

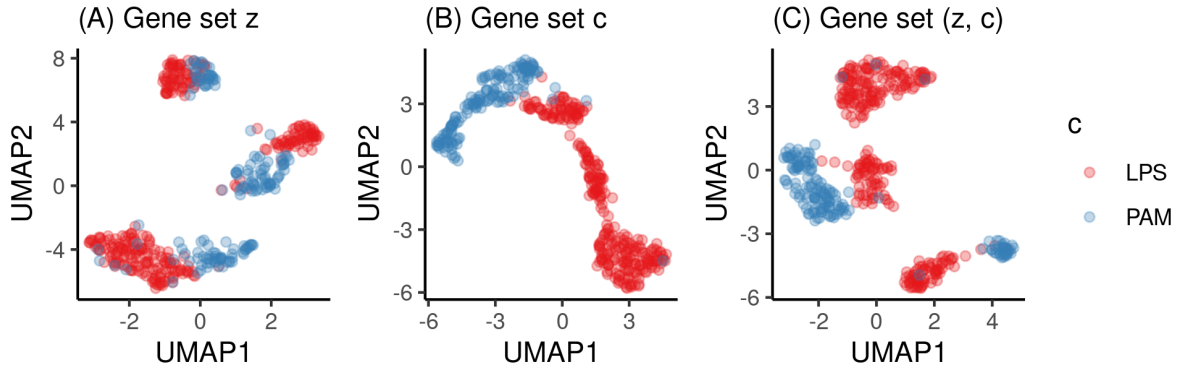

Figure 1: UMAP visualisations of gene sets with large (A) additive z, (B) additive c and (C) interaction effects from the mouse dendritic single cell data.

C Additional results for synthetic data

The below Table summarises results for a series of experiments, quantifying how accurately was the true underlying variance decomposition identified. We separate the (a) purely linear and (b) a more challenging mix of linear and non-linear data generative mechanisms. For linear dependencies, the linear-CVAE works well, but so does ND-CVAE despite the increased flexibility. However, when considering more challenging non-linear dependencies, ND-CVAE outperforms the linear-CVAE.

Table 1: Correlation between the true underlying variance component $\text{Var}(f)$ and the inferred value, measured on synthetic gene expression data, under (a) linear and (b) non-linear data generative mechanism.

	(a) linear		
	$\text{Var}(f_z)$	$\text{Var}(f_c)$	$\text{Var}(f_{zc})$
linear-CVAE	**0.99**	**0.87**	**0.92**
naive ND-CVAE	0.56	0.42	0.37
ND-CVAE	**0.97**	**0.88**	**0.92**
	(b) non-linear		
linear-CVAE	0.46	0.05	0.40
naive ND-CVAE	0.41	0.38	0.19
ND-CVAE	**0.98**	**0.45**	**0.56**

D Batch correction experiment

Here we devise a synthetic experiment generated from a two-dimensional latent space. The data consists of two batches where each feature is either unperturbed, differs by a constant by batch or varies with z_1 by batch. Our goal is two-fold:

- To learn a 2D latent space where $\mathbf{z}$ would be adjusted for the confounding batch effect (i.e. we want $\mathbf{z}$ not to be predictive of the batch label)

- To characterise the variance decomposition of every feature in terms of z_1, z_2 and c.

The goal was to identify if any tested VAE variant was capable of achieving *batch correction* (here c=[batch]) by identifying a latent space in which the two batches overlapped each other (Fig 2). .

With the restricted representation power of Linear-CVAE, only translational shifts in the latent space could be corrected. Surprisingly, the standard CVAE did not entirely remove the batch effect in the latent space either. However, the sparse ND structure within the ND-CVAE has correctly identified a (z_1, z_2) space in which the batches are now intermixed and the nonlinear batch effects removed. Furthermore, ND-CVAE lets us characterise how features vary with latent z_1, z_2 and known c.

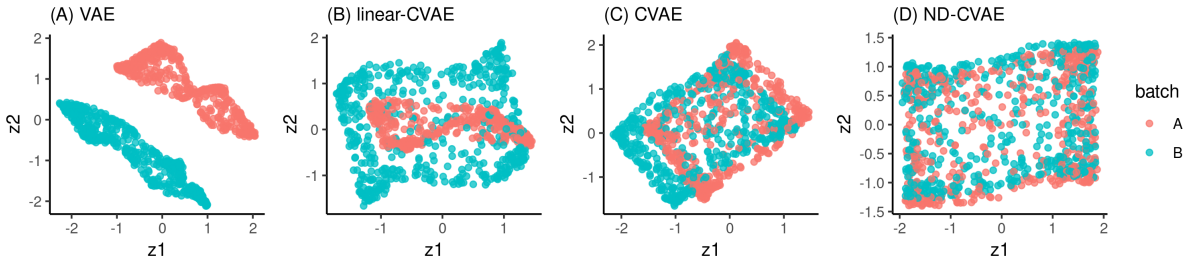

Figure 2: ND-CVAE recovers a batch-adjusted two-dimensional **z** on a synthetic example whereas other approaches (VAE, CVAE, and linear-CVAE) struggle to appropriately adjust for known .

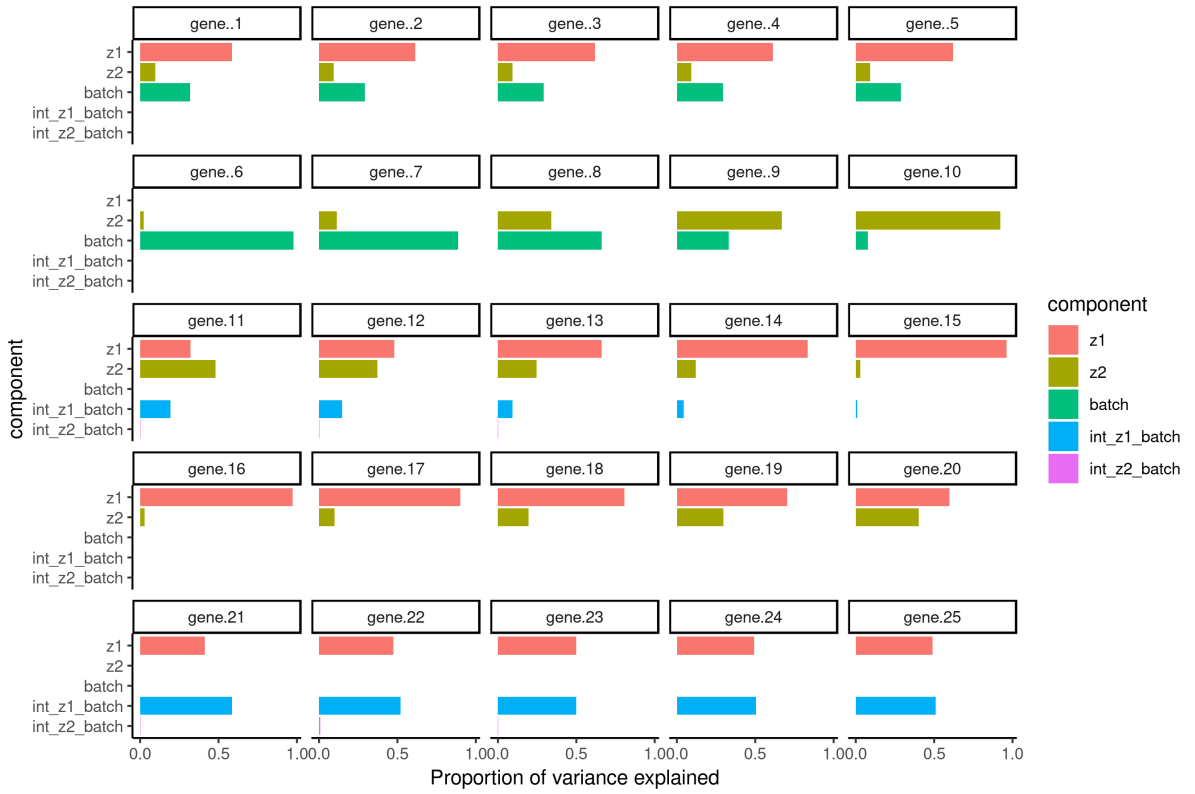

Figure 3: On the synthetic data set, we characterised the neural decomposition for every gene as a function of z_1, z_2, batch c, and interactions between them.

E Connection to a GP-based decomposition

Here we discuss how the Neural Decomposition behaviour relates to a conceptually similar c-GPLVM decomposition. The latter, being a decomposition of Gaussian Processes, is better understood in the sense that for any configuration of kernel hyperparameters, the integral constraints can be fulfilled analytically in closed form via conditioning. Thus, the GP decomposition is *exact* in the sense that all the integral constraints can be fulfilled exactly rather than approximately. However, the two model classes have different properties: they make different assumptions and have different scalability properties. We will discuss both of these below.

E.1 Enforcing integral constraints

Thus, in some sense, one could consider a GP-decomposition as a golden standard for this purpose, however note that the set of functions that have positive probability under the GP prior (e.g. under the squared exponential kernel these are infinitely differentiable functions) does not necessarily overlap with the set of functions that are parameterised by a neural network. The former is determined by the kernel, whereas the latter is determined by various neural architectural choices.

In Figure 4 we have investigated how the behaviour of Neural Decomposition differs from the GP-based functional decomposition on synthetic example that involves inference over f_z, f_c, and f_{zc}. We have visualised the inferred GP mean and the inferred ND mappings (using a one hidden layer architecture, with 64 neurons, either with a ReLU or Softplus nonlinearity) for both f_z (on the left) and f_{zc} (on the right), highlighting the L^1 distance between the two functions. For the ND with identifiability constraints (top row), this distance is relatively small and there are only minor differences from the GP posterior means, whereas the mappings inferred by the unidentifiable ND without constraints (bottom row) differ significantly from the GP ones.

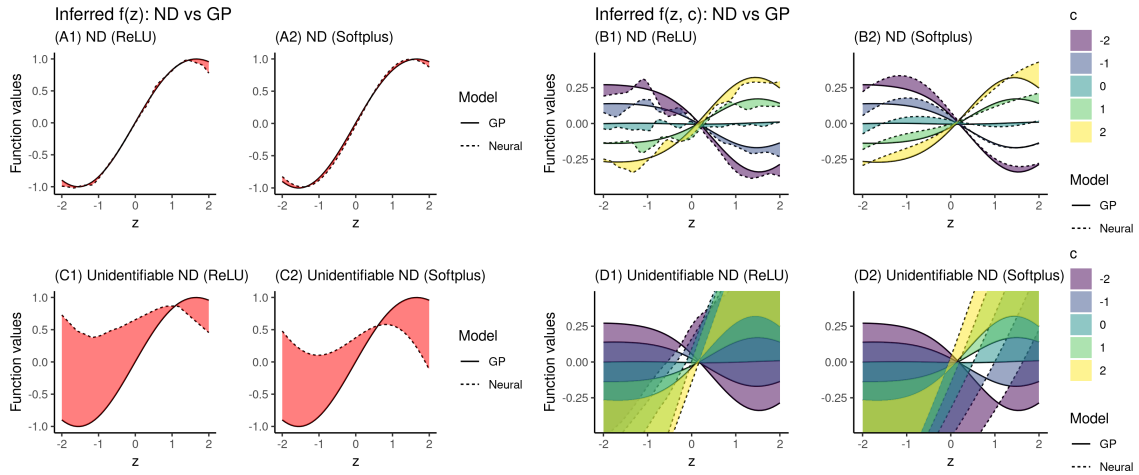

Figure 4: On a synthetic data set ($N = 500$) we compare the Neural Decomposition with the GP exact decomposition directly in the function space, visualising both the predicted mean for the inferred f_z (panels A) and f_{zc} (panels B, shown for $c \in \{-2, -1, 0, 1, 2\}$) as well as the L^1 distance (shaded area) between them. The functional subspace defined by ND depends on various architectural choices such as nonlinearity: we used ReLU in panels (A1, B1) and Softplus in (A2, B2). The ND without identifiability constraints (bottom row) learns a decomposition which lies far (in L^1 distance) from the GP one.

E.2 Computational considerations

Despite its elegant theoretical underpinnings, the GP-decomposition suffers from scalability issues intrinsic to GP-based models. While the cubic complexity w.r.t. sample size N can be addressed via inducing-point methods, the scalability of c-GPLVM w.r.t. data dimensionality P can become the limiting factor for high-dimensional data. While the decomposable c-GPLVM scales linearly with P both in terms of compute and memory, even for moderate P it becomes prohibiting to fit c-GPLVM on a laptop because of the memory requirements, as shown in Figure 5.

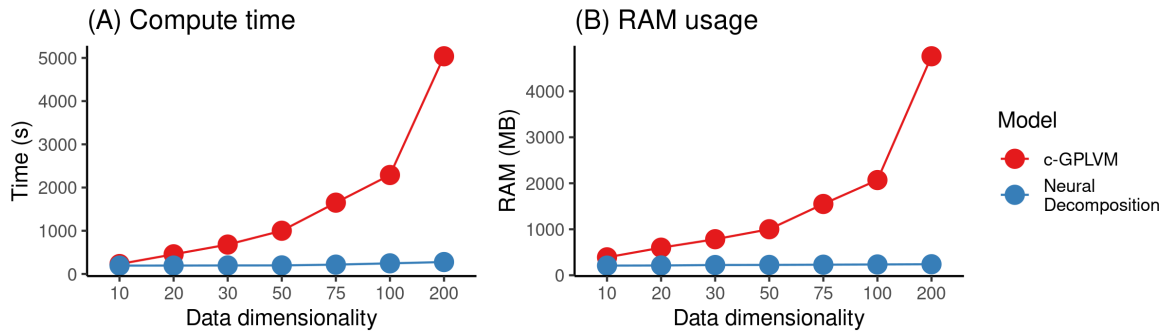

Figure 5: Neural decomposition is much more scalable w.r.t. data dimensionality than the GP decomposition (inducing-point implementation of the c-GPLVM), both in terms of compute time and memory requirements. For fixed sample size $N = 500$, we varied the dimensionality of data $P \in \{10, 20, \ldots, 100, 200\}$, and compared (A) the compute time, and (B) RAM usage for both methods when varying P (on x-axis). Experiments were made on a desktop with 8 Intel i7-6700 CPUs, both implementations in PyTorch.