The Deep Generative Decoder

MAP estimation of representations improves modelling of single-cell RNA data

Schuster, Viktoria; Krogh, Anders

Published in:
Bioinformatics

DOI:
10.1093/bioinformatics/btad497

Publication date:
2023

Document version
Publisher's PDF, also known as Version of record

Document license:
CC BY

Citation for published version (APA):
Gene expression

The Deep Generative Decoder: MAP estimation of representations improves modelling of single-cell RNA data

Viktoria Schuster and Anders Krogh

1 Introduction

High-throughput methods in biology and medicine produce vast amounts of high-dimensional noisy data that we seek to extract knowledge from. The first step is often to obtain a low-dimensional representation of the data through clustering, PCA analysis or other similar techniques. A good example is gene expression analysis, where we need to compare counts for each of the 10–20 thousand genes between samples or cells. Here, the first analysis is often to visualize the data in two dimensions with UMAP (McInnes et al. 2018) or t-SNE (van der Maaten and Hinton 2008). Although these tools are indispensable, they do not model the uncertainty of the count data or reveal any of the underlying structure in the data. Therefore, it has become popular to make use of generative models that give low-dimensional representations mapping to a probability distribution over the data. Some of the most common are variational autoencoders (VAEs) (Kingma and Welling 2013), generative adversarial networks (Goodfellow et al. 2014), normalizing flows (Rezende and Mohamed 2015), and energy-based models (Lecun et al. 2006). Even though these approaches have their shortcomings (Bond-Taylor et al. 2021), they have shown great success in a multitude of applications, such as single-cell RNA-sequencing (Lopez et al. 2018, Seninge et al. 2021), image and surface generation (Abukmeil et al. 2021, Kammoun et al. 2022), and natural language processing (Abukmeil et al. 2021, Wali et al. 2022). In addition, diffusion models (Ho et al. 2020) and Transformers (Vaswani et al. 2017) are among the best performing and most popular approaches in generative modelling of image and text data. However, these methodologies do not necessarily include probabilistic latent spaces.

The VAE is the most common type of generative model applied to biological data. In general, VAEs consist of an encoder that maps data to a lower-dimensional probabilistic representation in latent space and a decoder that maps back to the data. The model is trained using maximum likelihood estimation (MLE), which seeks to find a set of parameters that maximize the likelihood of the data. However, the likelihood calculation is intractable due to an integral over the latent space. The solution in the VAE is to approximate the likelihood and maximize a variational lower bound (the ‘ELBO’). This approximation is not always accurate (Cremer et al. 2018) and favours larger latent dimensionalities than necessary (Yacoby et al. 2020).

The decoder alone specifies the generative model and the encoder of the VAE is just a convenient tool for finding representations, but it adds a whole new set of modelling choices to be made, which influence the inference gap and expressiveness (Cremer et al. 2018). As a result, encoder-less representation learning has been suggested many times in different ways.
There are simple, deterministic approaches that maximize the likelihood over representations of the training data and the decoder parameters at the same time (Han et al. 2017, Bojanowski et al. 2018, Zadeh et al. 2021). Here and in our previous work, the representations are treated like the other model parameters and since gradients can be derived exactly, the model can be trained using standard gradient descent. There are also other encoder-less generative models, such as Gaussian Process Latent Variable Models (GP-LVMs) (Lawrence 2003) or PCA-based models (Collins et al. 2001, Mohamed et al. 2008, Townes et al. 2019). In their vanilla form, they have issues with scaling and limited model complexity and have not quite gained traction in the application to biological data compared to VAEs. However, scalable versions of GP-LVMs have been presented (Ahmed et al. 2019, Verma and Engelhardt 2020, Lalchand et al. 2022) and PCA-based methods have been applied for the joint modelling of multiple single-cell modalities (Mourragui et al. 2023). Scalability in these GP-LVM applications is achieved by applying sparse Gaussian processes and amortized inference (Ahmed et al. 2019, Verma and Engelhardt 2020, Lalchand et al. 2022). As a result, they are still limited to the same problems of variational inference (VI) (Cremer et al. 2018) and sensitivity of latent initialization (Lawrence 2003). The standard VAE and many other models use a fixed Gaussian distribution over latent space. This may result in under-represented representations, and it has been shown that learning the parameters of a Gaussian mixture leads to improved generalization and clustering (Dilokthanakul et al. 2017, Guo et al. 2020). However, this makes the VI even more complex, as it has to be combined with score matching (Vahdat et al. 2021), or other approaches. Another alternative approach worth mentioning is the VQ-VAE (van den Oord et al. 2017), which learns discrete representations with more powerful priors than a standard Gaussian, but still requires an encoder.

In this work, we present a probabilistic formulation of a generative decoder using maximum a posteriori (MAP) estimation of all parameters. MAP estimation is the Bayesian analogue of MLE and seeks to find the set of parameters that maximize the posterior, which is the probability of the representations and model parameters given the observed data. There is no intractable step in this estimation, so both model parameters and representations can be estimated directly. The model we are introducing consists of a decoder (or generator), such as a feed-forward neural network, and a distribution over representations with learnable parameters, which are optional. Estimating the representations of the training samples as well as the parameters is straight-forward and can be done by gradient descent as in a non-probabilistic formulation (Schuster and Krogh 2021a). One of the main advantages of this approach over VI is the ease and flexibility with which distributions over representations can be learned. The advantage over GP-LVMs is that the approach is scalable. Our emphasis here is on a model with a parameterized distribution over representations. This is motivated by the intuition that representations are meaningful and are likely to group samples with different properties. We think of this approach as a marriage of generative models and manifold learning, such as UMAP. The beauty of the approach lies in its simplicity and utility. The decoder parameters, latent representations and latent distribution are estimated in the same way. The simplicity makes it easy to extend the model to more complex distributions and losses. Unlike in Gaussian Mixture VAEs (Dilokthanakul et al. 2017b, Bai et al. 2022), no additional encoder for the Gaussian components is needed. It also makes it much simpler to understand for non-specialists. This is especially important, as we see great potential for this approach in fields like biology and medicine.

In the following sections, we present the theory of the model and experiments on two very different types of data. We first demonstrate the general functionality of the proposed model on the Fashion-MNIST (Xiao et al. 2017) benchmark dataset. This is a typical choice in deep learning and an important step, because it enables us to investigate and understand the model’s generative capabilities and potential caveats. We further explore the complexity of the distribution over representations and compare latent space and generative capabilities to models using VI. We also showcase the extent of the approach to supervised learning in which the mixtures of the latent distribution represent specific data classes. After establishing the model’s functionality, we show an application to single-cell gene expression count modelling. We chose a dataset of peripheral blood mononuclear cells (PBMC) (Zheng et al. 2017), for which we compare our approach in terms of data reconstruction and clustering performance to scVI (Gayoso et al. 2022) and scVAE (Grønbech et al. 2020). The latter is a VAE also using an adaptive mixture of Gaussians rather than a single fixed Gaussian to model the latent representation. We show that our model learns a representation and Gaussian mixture model which cluster well according to cell type with additional, previously unobserved sub-clustering. Finally, we compare the DGD and scVI on 11 different single-cell gene expression datasets using default settings of scVI and our model.

2 Materials and methods
2.1 Models
2.1.1 The DGD
Consider a model with observed variable $x$ and continuous latent variable $z$, which we also refer to as a representation. Usually, the $x$-space (or sample space) is of a higher dimension than the $z$-space, so the model gives a low-dimensional representation of data. A neural network, the decoder, with parameters $\theta$ takes $z$ as input and outputs $f_\theta(z)$, which are the parameters for the distribution of $x$. These could be the means and standard deviations of independent normal distributions. Thus, the neural network defines a conditional distribution in sample space, $P(x|z, \theta) = P(x|f_\theta(z))$. We assume a distribution over representation space, $P(z|\phi)$, with parameters $\phi$. The parameters may be fixed as in a standard VAE with a fixed Gaussian distribution over representations. It can also have adjustable parameters, such as a mixture of Gaussians with trainable means, covariances, and mixture coefficients, but essentially any (differentiable) parameterized distribution can be used. When introducing priors over parameters, the joint probability of everything becomes

$$P(x, z, \phi, \theta) = P(x, z|\phi, \theta)P(\phi)P(\theta)$$

Again, $P(x|z, \theta)$ represents the decoder neural network, $P(z|\phi)$ the distribution over representations, $P(\theta)$ the prior over the decoder parameters (neural network weights), and
The Deep Generative Decoder

Figure 1. The model. (a) Graphical model and (b) schematic of the deep generative decoder. The DGD consists of a decoder of any desired architecture with parameters \( \theta \) mapping the latent representation \( Z \) to the data space \( X \). The representation is modelled by a probability distribution with parameters \( \phi \). \( N \) is the number of samples.

\[ P(\phi) \] the prior over the parameters in the representation space distribution. The model schematic is shown in Figure 1. We use MAP estimation to find optimal parameters and representations. For a dataset \( X = \{x^1, x^2, \ldots, x^N\} \) with \( N \) samples we thus want to find representations \( Z = \{z^1, z^2, \ldots, z^N\} \) and parameters \( \phi, \theta \) that maximize \( P(Z, \phi, \theta | X) \):

\[
\begin{align*}
\arg\max_{Z,\phi,\theta} P(Z, \phi, \theta | X) &= \arg\max_{Z,\phi,\theta} P(X | Z, \phi, \theta) P(Z, \phi) / P(X) \\
&= \arg\max_{Z,\phi,\theta} P(Z, \phi) \\
\end{align*}
\]

If we assume independence among training samples, the log of this joint probability can be obtained from (3), with \( i \) indexing the samples

\[
\log P(X, Z, \phi, \theta) = \sum \left( \log P(x^i | z^i, \theta) + \log P(z^i | \phi) \right) + \log P(\phi) + \log P(\theta).
\]

This is the quantity we want to maximize with respect to both the representations \( (z) \) for all samples and the model parameters \( (\phi, \theta) \). The training process can be imagined like this: a representation \( z \) for each training sample is initialized. This can be a random sample from a chosen distribution or a zero-valued vector. Representations are then passed through the decoder to give \( P(x | z, \theta) \) and losses are computed from this and \( P(z | \phi) \). The decoder can be any type of Neural Network that fulfills this constraint. The choice of reconstruction loss is equally flexible but should ideally represent \( P(x | z) \). All parameters \( \theta, \phi, \) and \( z \) are then updated via backpropagation. The implementation is efficient and straightforward and compatible with standard modules and loss functions in deep learning frameworks, such as PyTorch.

Algorithm 1 summarises the process.

Since representations are updated once per epoch and decoder and GMM parameters every batch pass, we use different instances of the optimizer for the different parameter sets. We have also observed that the GMM requires much larger steps than neural network parameters, so we choose to have an optimizer per parameter set with individually selected learning rates.

Once the model is estimated, prediction on new data points is done by first finding an optimal representation by maximizing \( P(x | z, \theta) P(z | \phi) \) as above while keeping all other model parameters fixed, i.e. do gradient descent in \( z \) alone. For each new data point, one or more new representations are initialized. The type of initialization can be chosen. It can be from zero as done for the training data or from component means. For the latter initialization technique, the best starting point for each new sample is derived from the minimum reconstruction loss from all \( K \) component means. This presents our default setting and essentially assigns the optimal component to each new sample. From there on, representations are typically optimized for a very short time (10 epochs) with a batch size of 32. Inference of new representations is of course done with frozen decoder and GMM parameters.

In this work, we use a parameterized Gaussian mixture for the representations \( P(z | \phi) \) with a mollified Uniform (‘softball prior’) as prior over component means. Hereafter, we will refer to this Gaussian Mixture Model as GMM. The mixture model consists of \( K \) mixture components each with a mean vector \( \mu \) with same dimensionality as the

\[
\begin{align*}
\text{Algorithm 1. Training} \\
\text{Initialize parameters for representations } z^i, \text{ decoder and GMM} \\
\text{for epoch in } n_{epochs} \text{ do} \\
\text{for } x^i, i \text{ in training data do} \\
\quad z^i = Z, \\
\quad y^i = \text{model}(x^i) \\
\quad \text{loss} = L_{\text{reconstruction}}(y^i, x^i) + L_{\text{GMM}}(x^i) \\
\quad \text{Backpropagation} \\
\quad \text{Optimizer step for model and GMM} \\
\quad \text{end for} \\
\quad \text{Optimizer step for Representation} \\
\text{end for}
\end{align*}
\]
representations \((m)\), diagonal covariance \(\Sigma\) and a mixture coefficient \(c\) for each component. The mixture coefficients are transformed into component weights \(w\) through softmax activation. Since the diagonal covariance matrix can only take positive values, the according parameter is learned as the negative log-covariance.

We use a weight decay in the training of the decoder, which corresponds to a Gaussian prior on the weights. The priors over the GMM parameters \(\phi\) are as follows. The prior on the mixture weights is an \(m\)-dimensional Dirichlet distribution with uniform parameters, \(\alpha\). The prior on mixture means is a mollified uniform which we call the ‘softball’ distribution with hyperparameters scale as the radius of the \(m\)-ball and sharpness determining the slope of the boundary. Details on initialization and formulation of the log-probability can be found in Supplementary section ‘The softball prior’. For the negative log-variance, we use a Gaussian prior \(\mathcal{N}(-2 \log(\sigma), 1)\) with the same mean and standard deviation for all dimensions. Mixture coefficients and negative log-variances are initialized with default values 1 (so that components weights are uniformly \(K^{-1}\)) and \(-2\log(\sigma)\) with \(\sigma = 0.2 \times \text{scale} \times K^{-1}\), respectively. The default values for \(\alpha\), scale, and sharpness are all 1. Component means are initialized by sampling from the softball prior. The distributional component of the loss is given as

\[
-\log P(z) = -\log \left( \sum_k \exp \left( w^k \sum_i \log P(z^i | \phi^k) \right) \right) \\
-\log P(\phi) \\
\text{with } P(z^i | \phi^k) = P(z^i | \mu^k, \sigma^k, -\log \Sigma^k)) \\
\text{and } P(\phi) = P(\mu, \sigma, -\log \Sigma).
\]

with \(P(z^i | \mu^k, \sigma^k, -\log \Sigma^k)\) as the density of the \(k\)th multivariate Gaussian component for representation \(z^i\).

2.1.2 Fashion-MNIST DGD

For the image-generating DGD, we choose a network architecture based on convolutions and tricks from other image generation models and image segmentation techniques (He et al. 2016, van den Oord et al. 2016, Vahdat and Kautz 2020). The decoder starts with two fully connected hidden layers fed with the latent representations. The first hidden layer has 100 units, the second \(\text{capacity} \times 3 \times 3\) units. The capacity represents the minimum number of channels of the convolutional layers except for the grey-scale output channel and is set to 64. The output of the fully connected hidden layers is reshaped into \((\text{batch}, 3, 3, \text{capacity})\) and fed into a series of NN blocks. These blocks consist of a transposed convolutional layer, Swish activation and a Squeeze and Excitation layer as applied by Vahdat and Kautz (2020). Input- and output channels as well as kernel size, strike, and padding depend on the position of the block in the scheme and can be seen in Supplementary Fig. S5. There are four main blocks and two skip connection blocks. The outputs of combined blocks go through the activation function after summation. The series of these NN blocks is followed by a PixelCNN with five layers and mask size five (van den Oord et al. 2016) and a last simple transposed convolutional layer reducing the number of channels to one. The output is scaled using Sigmoid activation.

Latent dimensionality and convolutional capacity vary depending on the experiment. For the models with 10 and 20 Gaussian components, we use a latent dimension of 20. The standard deviation of the GMM components are calculated as one. This ensures that the components are sufficiently separated and alleviates the burden of another hyperparameter to optimize. The scale and hardness of the softball mean prior are three and five, respectively, and the Dirichlet alpha is set to two since we are dealing with balanced classes. For the models involving a standard Gaussian (including VAD and VAE), we test latent dimensionalities of 20, 50, and 100. Since the prior is not learned, softball and Dirichlet prior are not relevant. The Supplementary material contains results of a model with latent dimension 2, capacity 32, and varying number of Gaussian components (1, 10, and 20).

For training, we use the binary cross-entropy (BCE) loss and Adam optimization with betas 0.5 and 0.7. For decoder, representation, and GMM, we choose learning rates \(1e^{-3}, 1e^{-2}, 1e^{-1}\), respectively. This relationship has evolved as a rule of thumb for the DGD. Learning rates are best chosen with respect to a desired decoder learning rate (which is to be optimized for each task independently). From there, the representation learning rate should be 10 times the decoder learning rate, and the GMM learning rate should be between 10 and 20 times that of the decoder.

Hyperparameters, such as the number of hidden dimensions, capacity, dropout, and the softball prior scale were found through optimization with respect to the validation reconstruction loss. The summary is available as a parallel coordinate plot from weights and biases runs in Supplementary Fig. S1.

2.1.3 VAD and VAE

The VAD and VAE decoder implementations tested here are architecturally identical to the DGDs with a standard Gaussian. The VAE encoder was for simplicity chosen to mirror the decoder, with normal convolutional layers instead of the transposed convolutions. The capacity for all models is 32 and latent dimensions tested are 20, 50, and 100. Instead of the negative log-density of the GMM, the loss for the distribution over latent space is given as the Kullback–Leibler divergence as typical for VI (Kingma and Welling 2013).

2.1.4 Supervised learning

In the unsupervised model, the log-likelihood \(\log P(z^i | \phi)\) of an individual representation \(z^i\) being drawn from the parameterized distribution \(P(\phi)\) is given by the log sum of the probability densities of \(z^i\) per component times the corresponding component probabilities, which are given by the softmax of the weights. In the supervised setting, we can ignore all components except the one that has been assigned to the sample’s class. We thus only have to calculate the probability density of \(z^i\) for the given component multiplied with the component probability. The losses for all other components will be zero and they will thus not receive gradients.

2.1.5 Single-cell DGD

The architecture of scDGD is much simpler than that of the Fashion-MNIST DGD. The decoder consists of the input layer with units defined by the latent dimensionality, and three hidden layers of each 100 units, connected to the output layer of (in the case of the PBMC dataset) 32 728 units, representing all transcripts in the data. The number of output units is given by the number of genes with non-zero expression counts in the data. The latent dimensionality was empirically found
best to be 20 in terms of validation reconstruction loss and clustering accuracy of the cell types. The depth of the network was also empirically determined. The parallel coordinate summary of the hyperparameter optimization can be found in Supplementary Fig. S6.

The normalized expression counts are modelled with a Negative Binomial distribution. Normalization is achieved by dividing the true expression count with the sample’s largest count. We can therefore use Sigmoid activation on the output layer. The reconstruction loss is given as the probability density of the re-scaled expression count and a gene-specific, learned dispersion parameter. Since this parameter is positive definite, we learn its logarithmic counterpart. For training, we use Adam optimization with betas 0.5 and 0.7. For decoder, representation, and GMM, we choose learning rates $1 \times 10^{-3}$, $1 \times 10^{-2}$, and $1 \times 10^{-2}$, respectively. Two models are trained with 9 and 18 Gaussian components, respectively, for the PBMC dataset.

The component means of the GMM are distributed according to the mollified Uniform, for which we set scale and sharpness to one each. This applies to both 9-component and 18-component model. The standard deviation of the components is initialized with a mean of 0.02 for the 9-component model and 0.01 for the 18-component model, which roughly corresponds to the formula we have introduced in the previous section, but is modified to a fifth of that in order to improve the component separation. Dirichlet alphas are set to two and one for models with 9 and 18 components, respectively. This is motivated by a Dirichlet alpha of one allowing for non-uniform component weights, which is desired in this case of imbalanced cell type classes. In the case of nine Gaussian components, a Dirichlet alpha of one resulted in even more components representing T cells, so we decided to distribute the components more uniformly by increasing alpha.

All hyperparameters described above, except for the number of GMM components and of course the output dimensionality, present the empirically determined default parameters of scDGD. We applied this default model to 11 more datasets, in which the output dimensionality was set as the number of transcript found in the data, and the number of Gaussian components was determined by the number of unique cell types identified by CellTypist. Models for all datasets except PBMC (20k) were trained for 800 epochs. PBMC (20k) was trained for 1000 epochs. In all scDGD models, the learning rate of the decoder was reduced to $1 \times 10^{-4}$ after 500 epochs.

### 2.1.6 scVI

An scVI model was trained on our PBMC train set. The model is implemented in Python version 3.8 with the scvi-tools (Gayoso et al. 2022) package version 0.17.4, as described in Lopez et al. (2018) with one hidden layer of 128 hidden units in both encoder and decoder. We chose a latent dimension of 20 for comparability with our model. The dropout rate is set to 0.1. The model was trained for 1000 epochs. For the analysis and comparison to scDGD, the latent space is clustered with $K$-means clustering ($K = 9$) and then evaluated by the ARI. The lower bound is computed on our held out test set. Latent representations and ELBO were computed using scVI’s implemented functions. NLLs and RMSEs were computed based on the models returned mean and dispersion parameters for the test set.

For the remaining datasets, the scVI models were again initialized with default parameters and trained for up to 400 epochs, which represents the upper bound of the default number of epochs (for large datasets, the automatically calculated number of epochs can be very low).

### 2.2 Datasets

In this work, we made use of in total 13 publicly available datasets, out of which 12 represent single-cell transcriptomics sets. The dataset referred to as PBMC was used to develop the application of the DGD to single-cell expression data. The remaining single-cell datasets were used to evaluate the performance of scDGD with default parameters in comparison to the popular and successful scVI model with default parameters.

#### 2.2.1 Fashion-MNIST

The dataset used in our proof of concept is the natural image Fashion-MNIST (Xiao et al. 2017) set. We used the dataset’s implemented train-test split.

#### 2.2.2 PBMC

The dataset used to develop the single-cell application of our model is a single-cell gene expression count dataset of PBMC presented by Zheng et al. (2017). The data are provided by 10× Genomics under ‘Single Cell 3’ Paper: Zheng et al. 2017 (v1 Chemistry)’ and consists of data from the following nine cell types: CD4+/CD45RA+/CD25- naïve T cells, CD4+ helper T cells, CD4+/CD25+ regulatory T cells, CD4+/CD45RO+ memory T cells, CD8+/CD45RA+ naïve cytotoxic T cells, CD8+ cytotoxic T cells, CD56+ natural killer cells, CD34+ cells, and CD19+ B cells. As Grønbech et al. (2020), we used the filtered gene-count matrices. These data are extremely sparse with 98% of the data being zero (Grønbech et al. 2020). Of the 32,738 genes covered by this assay, only 21,812 are expressed in the whole dataset at least once. The 92,043 samples are randomly split into train, validation, and test set using percentages 81%–9%–10%. The validation set is used to finding hyperparameters, and the test set is used for final evaluation.

#### 2.2.3 Single-cell evaluation sets

The following datasets were taken from 10× Genomics or chosen because of their use in the scVI paper (Gayoso et al. 2022). We used provided raw counts of cells that passed quality control filtering and did not apply any feature selection, modelling all transcripts available. Datasets were annotated using the CellTypist (Dominguez Conde et al. 2022, Xu et al. 2023) python package. The resulting cell type labels were used in the DGD for automatic selection of the number of Gaussian components and in the clustering performance evaluation as an approximated ground truth. All datasets were split into 80% training, 10% validation, and 10% test data. The raw counts with cell type and data split assignment as observables were exported as AnnData (Virshup et al. 2021) objects, which were used to train and evaluate both scDGD and scVI.

**PBMC (500)**

The smallest of the datasets used for the elaborate analysis of our model’s performance and applicability is the ‘PBMCs from human (3’LT v3.1, Chromium X), Single-Cell Gene Expression Dataset by Cell Ranger 6.1.0 (2019)’ dataset retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/500-human-pbm-cs-3-lt-v-3-1-chromium-x-3-
It contains 587 samples and 36 601 features. For cell type annotation, we used the ‘Immune_All_Low’ model as reference and majority voting in CellTpyist. This resulted in the presence of 11 distinct cell types.

BMMC (2k)
The next dataset, ‘Frozen BMMCs (Healthy Control 1), Single-Cell Gene Expression Dataset by Cell Ranger 1.1.0 (2016)’, contains 1985 bone marrow mononuclear cells with 32 738 features and was retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/frozen-bmm-cs-healthy-control-1-1-standard-1-1-1-1-4. Cell type annotations were approximated using CellTpyist with the ‘Immune_All_Low’ reference model and majority voting. This resulted in 15 distinct cell types.

Cortex (3k)
A dataset of 3005 cells from mouse cortex and hippocampus with 19 972 features was published in Zeisel et al. (2015). Cell type annotations were approximated using CellTpyist with the ‘Developing_Mouse_Brain’ reference model and majority voting. This resulted in 14 distinct cell types.

Jejunum (5k)

Mouse brain (5k)
This dataset ‘5k Adult Mouse Brain Nuclei Isolated with Chromium Nuclei Isolation Kit, Single-Cell Gene Expression Dataset by Cell Ranger 7.0.0 (2022)’ comprised 7377 cells from the adult mouse brain with 32 285 features and retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/5k-adult-mouse-brain-nuclei-isolated-with-chromium-nuclei-isolation-kit-3-1-standard contained 4392 cells with 36 601 features. Cell type annotations were approximated using CellTpyist with the ‘Developing_Mouse_Brain’ reference model and majority voting. This resulted in 24 distinct cell types.

Whole blood (8k)
From the blood of healthy females, a total of 8000 PBMCs, Neutrophils and Granulocytes were extracted into the dataset ‘Whole Blood RBC Lysis for PBMCs and Neutrophils, Granulocytes (3‘), Single-Cell Gene Expression Dataset by Cell Ranger 6.1.0 (2021)’, containing 36 601 features. It was retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/whole-blood-rbc-lysis-for-pbmc-neutrophils-granulocytes-3-3-1-standard. Cell type annotations were approximated using CellTpyist with the ‘Immune_All_Low’ reference model and majority voting. This resulted in 10 distinct cell types.

Heart (10k)
The 7713 cells with 31 053 features in ‘10k Heart Cells from an E18 mouse (v3 chemistry), Single-Cell Gene Expression Dataset by Cell Ranger 3.0.0 (2018)’ were extracted from an E18 mouse. The data were retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/10-k-heart-cells-from-an-e-18-mouse-v-3-chemistry-3-standard-3-0-0. Cell type annotations were approximated using CellTpyist with the ‘Immune_All_Low’ reference model. This resulted in three distinct cell types.

PBMC (10k)
‘10k Human PBMCs [(3’LT v3.1, Chromium X (with intronic reads)], Single-Cell Gene Expression Dataset by Cell Ranger 6.1.2 (2022)” retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/10-k-human-pbmc-3-v-3-1-chromium-x-with-intronic-reads-3-1-high contains 11 984 PBMCs from healthy female donors between the ages of 25 and 30, with 36 601 transcripts covered. Cell type annotations were approximated using CellTpyist with the ‘Immune_All_Low’ reference model and majority voting. This resulted in 19 distinct cell types.

PBMC (20k)
Another dataset from female donors aged 25–30 provided was used, containing a total of 23 837 cells with 36 601 features. The ‘20k Human PBMCs (3’ HT v3.1, Chromium X), Single-Cell Gene Expression Dataset by Cell Ranger 6.1.0 (2021)’ data were retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/20-k-human-pbmc-3-hv-3-1-chromium-x-3-1-high-6-1-0. Cell type annotations were approximated using CellTpyist with the ‘Immune_All_Low’ reference model and majority voting. This resulted in again 19 distinct cell types.

Hemato (25k)
After excluding the ‘basal-bm1’ library due to poor quality [as presented in Gayoso et al. (2022) by recommendation of the authors], the dataset of hematopoietic progenitor cells from mice (Tusi et al. 2018) comprised 25 050 cells and 28 205 features. Cell type annotations were approximated using CellTpyist with the ‘Cells_Intestinal_Tract’ reference model and majority voting if cell types represented less than 100 cells. This resulted in 18 distinct cell types.

Mouse brain (1M)
The extremely large dataset of 1 306 127 brain cells from two E18 mice [‘1.3 Million Brain Cells from E18 Mice, Single-Cell Gene Expression Dataset by Cell Ranger 1.3.0 (2017)’] was retrieved from 10× Genomics, https://www.10xgenomics.com/resources/datasets/1-3-million-brain-cells-from-e-18-mice-2-standard-1-3-0. It contains samples from cortex, hippocampus, and the subventricular zone and features 27 998 transcripts. Cell type annotations were approximated using CellTpyist with the ‘Developing_Mouse_Brain’ reference model and majority voting. This resulted in 27 distinct cell types.

2.3 Performance metrics
2.3.1 Clustering metric
We use the Adjusted Rand Index (ARI) (Hubert and Arabie 1985) for evaluating and comparing clustering performance of our models. The value of the metric ranges from zero to one, representing random and identical clustering, respectively. This metric is relatively robust to different clustering approaches and diverging numbers of clusters, which makes it a good metric for comparing our method to others.
When analysing models without GMM priors, the latent spaces are clustered using $k$-means (Lloyd 1982) clustering implemented in Python’s scikit-learn package.

### 2.3.2 Reconstruction metrics

For Fashion-MNIST, we use the BCE loss in order to evaluate the reconstruction of image pixels.

In scDGDs, we calculate the NLL as the summed log-density of parameterized Negative Binomials over all genes. For each gene, the log-density is calculated based on the re-scaled model output and a gene-specific, learned, dispersion factor.

### 2.3.3 Image generation evaluation metric

For a quantitative evaluation of generated images, we employ the Fréchet Inception Distance (FID) (Heusel et al. 2017). The FID gives the squared Wasserstein distance between two multivariate Gaussian distributions. We used the PyTorch implementation (Seitzer 2020) based on the original publication Heusel et al. (2017). We compute the FID score three times for a given model, with random seeds 0, 21, and 87 243. In each run, we randomly generate 10,000 samples and compute the FID score with respect to the Fashion-MNIST test set.

### 2.4 Software, statistics, and visualization methods

Python 3.8 is used as the programming basis for all methods. Neural Networks are implemented in Pytorch 1.10 (Paszke et al. 2019) and training progress was monitored and logged using wandb (Biewald 2020). Dimensionality reduction is either done with our described model or common methods, such as PCA or UMAP (McInnes et al. 2018). If not stated otherwise, default parameters of 15 neighbours and a minimum distance of 0.1 are used in UMAP fits. Graph visualizations are achieved using the NetworkX package (Hagberg et al. 2008). Requirements can be found in the corresponding repository.

### 2.5 Hardware

Models were trained on NVIDIA TITAN Xp (12 GB), except for scVI on the mouse brain (1M) dataset, which was trained on NVIDIA TITAN RTX (24 GB).

### 3 Results

#### 3.1 Demonstration on image data

It is customary to test generative models on image benchmarks, so we begin our experimental analysis by applying our model to the Fashion-MNIST data. It allows us to understand and evaluate model behaviour, latent space and most importantly sampling quality much better than more complex, biological data. The decoder used for the Fashion-MNIST models is based on the DCGAN (Radford et al. 2016) generator architecture with modifications using common methods in image generation and is described in detail in Section 2 and Supplementary Fig. S5. We limited the design space (Radosavovic et al. 2020) to a latent dimension of 20, as interpolation can only occur in low-dimensional representations (Balestriero et al. 2021). The remaining parameters and how they were chosen are described in Section 2. The model was trained for 500 epochs.

In Fig. 2b, we see that a GMM with 20 Gaussian components distributes well over the complex latent space. The resulting distribution over latent space (the parameterized GMM) models the latent representation much better than for GMMs with less components which fail to pick up on more subtle structures (Supplementary Fig. S2). This supports the hypothesis stated in many works that overly simple distributions, such as standard Gaussians may not be sufficient for describing the probabilistic representations of many data.

Another advantage of the GMM is the increase in control over sampling. Figure 2a shows generated samples from individual components of the GMM. Having more components leads to each of them covering a much smaller area of the latent space, and thus offering less varied but more specific samples. In this case of a 20D latent space with 20 mixture components, each component mean can be attributed to a (sub-)class of Fashion-MNIST. A relatively high number of components can thus enable more deliberate, more qualitative sampling, and a better model of the latent space. While the latent representations in Fig. 2b show clear separation for super-classes, such as shoes (Ankle boot, Sneaker, and Sandal) and some distinct classes like of trousers and bags, other classes are much more mixed. We evaluate the clustering with the adjusted Rand index (ARI) (Hubert and Arabie 1985). This metric achieves values between 0 and 1 with 1 representing a perfect clustering and is corrected for chance. The ARI of this clustering based on predicted labels derived from the GMM’s component probabilities is only 0.295. As an extension of the DGD, it is also possible to perform supervised training, where each GMM component is assigned a data class a priori. In the optimization, each component is only committed to covering the latent space of its assigned classes’ representations. When trained with these assigned 10 components, we arrive at the distinct latent clusters seen in Fig. 2c with an ARI of 0.995 (which is merely a sanity check, since the ARI is expected to be high). While the quality of random samples from this model is reduced slightly (Supplementary Fig. S3) in comparison to those from an equivalent unsupervised model, we achieve perfect control over the sample identity.

We next investigated how the MAP approach compares to VI and amortization. For this purpose, we limit the prior of the DGD to a fixed standard Gaussian. We found that once the encoder of a VAE is removed [using the variational autoDecoder (VAD) (Zadeh et al. 2021)], reconstruction performance improves significantly and is equal to that of the DGD, as seen in Fig. 2d, Table 1, and Supplementary Table S1. The latter also shows that latent space normality is improved by removing the encoder. This supports Schuster and Krog (2021a), Bojanowski et al. (2018), and Bond-Taylor and Willcocks (2021) in their results stating that the encoder can be a hindrance to achieving good representations. Given the standard Gaussian priors, the structure of the latent spaces is not practically changed between the three models (Supplementary Fig. S4), even though ARIs vary a lot (Supplementary Table S1).

Our demonstration of the DGD on image data comes in very handy when next evaluating the generative modelling capabilities of the different approaches. The benefit of image data is that it comes naturally to us to evaluate whether generated data is sensible or not from a qualitative standpoint. In addition, there are many quantitative measures available. We make use of the Fréchet Inception Distance (FID) (Heusel et al. 2017), a popular metric of the similarity between image distributions. It is, however, a biased metric depending on the number of samples and the generator itself. A qualitative
analysis of the samples from all models (Fig. 2e) ranks the DGD samples the highest in terms of realisticness and detail. However, the FID score of the VAE samples are slightly better (Table 1) despite the generated images being very generic and blurry.

The FID score of the DGD improves drastically with learning a more complex parameterized distribution, which lends itself naturally to our approach. When using 20 Gaussian components, the FID decreases to 27.25 ± 0.11 (from three computations using different random seeds, see Section 2), positioning itself between the probabilistic autoencoder (28.0) (Bohm and Seljak 2022) and PeerGAN (21.73) (Wei et al. 2022).

### 3.1.1 Application to single-cell data

In order to assess the model performance on scRNA data, we chose to apply it to the PBMC data from Zheng et al. (2017).
The single-cell expression counts are modelled by a Negative Binomial distribution. We achieved best clustering results with a latent dimension of 20 and 3 fully connected hidden layers. The model, which we will refer to as single-cell DGD (scDGD), was trained for 700 epochs after which it achieved its overall best performance in terms of reconstruction and clustering. The number of GMM components was set to nine, which represents the number of cell types in the data. Details about our implementation and how we arrived at these hyperparameters can be found in Section 2 and in the Supplementary Material.

The resulting latent space is depicted as a UMAP projection in Fig. 3. From a visual perspective, the latent space and GMM components cluster well with the cell types. For clustering purposes, a cell is assigned to the component that gives the largest probability to its representation. The ARI of this clustering on the train set is 0.618. In the work of Grønbech et al. (2020), scVAE was reported to have a higher clustering

![Figure 3](https://example.com/figure3.png)

**Figure 3.** Latent spaces of scDGD. The latent spaces are shown coloured by cell type (left) and type of latent point (right). Visualizations are achieved using UMAP with a spread of 5 and a minimum distance of 1. ‘Representation’ refers to learned representation of training data, ‘samples’ correspond to random samples drawn from the GMM and ‘gmm means’ represents the component means. The IDs of the component means are shown in text on their corresponding coordinates. (a) Latent space of scDGD with nine Gaussian components. This model was trained for 700 epochs. (b) Latent space of scDGD with 18 components, trained for 600 epochs. (c–e) Spatial relationships between GMM components of the 18-component scDGD. (c) Heatmap of the GMM’s component assignment to samples given as the percentage of each class with highest probability of a component. (d) Graph visualization of the GMM component means with edge lengths correlated with Euclidean distances from (a) and edge widths negatively correlated with Euclidean distances. Only the 95th percentile of distances were accepted as graph edges, resulting in a threshold of 0.14. Edge colours show components belonging to the same cell type [same colours as in (b)], except blue, referring to all CD4 cells minus memory T cells. Edge widths are correlated with proximity of clusters.
performance than scVI of around 0.66 compared to 0.53 for scVI. However, scVI clearly outperformed the Gaussian Mixture VAE in terms of reconstructions. Since the scVAE tool could not be retrained, we as well compare our model to scVI and include the Leiden algorithm (Traag et al. 2019) as a baseline for clustering performance. The clustering performance of scVI is evaluated based on K-means clustering of the latent space with nine components.

As a baseline for clustering performance, we apply the Leiden algorithm commonly used in the field of single-cell data. At nine components, this achieves an ARI of 0.51. A default scVI model trained on our train set resulted in a clustering performance of 0.566, which is derived from K-means clustering with nine components. The reconstruction performance of scVI is on par with our model trained with nine components in terms of goodness of fit, but under-performs in reconstruction accuracy (Table 2). The reconstruction performance is evaluated based on two metrics. Firstly, we compute the negative log-likelihood (NLL) of the true held-out test counts being drawn from the gene-specific Negative Binomial distributions. These distributions are parameterized by the models’ predicted means and learned dispersion parameters. Secondly, we also compute the root mean squared errors (RMSE) as a standard metric for comparability. We include both metrics for a more comprehensive understanding of the reconstruction performance. While the NLL measures the goodness of fit of the probabilistic modelling of counts and leaves more flexibility for genes that are highly variable, the RMSE emphasises the reconstruction accuracy and highlights sensitivities to outliers in the modelling. The clustering of scVI derived from the K-means algorithm with nine components, however, is still lower than that of scDGD (Table 2). This shows that with scDGD, there is no need for a compromise with respect to built-in, high-performance clustering and count modelling.

We also trained a model with 18 Gaussian components, since we have learned in our explorations on image data that increasing the complexity of the distribution over representation can be beneficial to modelling the substructures in the representation if the problem is complex enough. Increasing the number of components resulted in equal reconstruction performance as seen in Table 2 and highly improved clustering performance with an ARI of 0.684. As we can see in Fig. 3c, several cell types have one or more components specifically assigned to them. Among these are CD4+ naïve and memory T cells, NK, CD34+, and B cells. The cytotoxic T cell sub-types cannot easily be differentiated by component assignment, but are distinct from all other cell types in the data. The same goes for all CD4+ T cells except memory cells. Since UMAP distorts the latent space, we also show a graph of the component means in Fig. 3d based on the Euclidean distances between them (Supplementary Fig. S7).

To go one step further in the analysis of our model, we next aimed to get an understanding of the structuring of the learned latent space. In both models with 9 and 18 components, we see that the CD34-positive cells form sub-clusters that are modelled by different Gaussian components. We therefore investigated the representations for CD34-positive cells learned by the 18-component scDGD in terms of some more fine-grained differentiation markers. We were able to determine a sub-cluster best modelled by component 17 (Fig. 4a), which is primarily occupied by cells expressing markers for the lymphoid lineage of haematopoietic stem cells (HSCs). This cluster further shows a distinction between bone marrow common lymphoid progenitors (CLPs) and cord blood CLPs, indicated by markers CD10 and CD7, respectively. Another sub-cluster, represented by components 8 and 16, is made up of several cells expressing CD18, which is a marker for myeloid cells regulating neutrophil production and a general marker for granulocytes. There is also a strong presence of CD14-positive cells in these clusters, suggesting an aggregation of monocytes and neutrophils. The biggest sub-cluster, modelled by component 14, suggests the presence of different sub-populations of the myeloid lineage of HSCs. We find gradual levels of CD41 to CD62L, which are markers for megakaryoblasts and myeloblasts, respectively. These are also correlated with component 7. We additionally found a concentrated site showing expression of different dendritic cell markers (CD1c, CD141, and CD303), best represented by components 8 and 16.

Lastly, we investigated the merits of scDGD in terms of user experience and applied both default scDGD and default scVI to 10 datasets that had no part in the development of scDGD. Details about these datasets from mouse and human and their preprocessing can be found in Section 2. Figure 5 shows that scDGD outperforms scVI in both data reconstruction (accuracy measured by RMSE) and latent clustering (in 9 out of 10 and 7 out of 10 cases, respectively). We also trained scVI and scDGD on the extremely large mouse brain dataset with roughly 1.5 million cells from 10× Genomics (see Section 2). On this large dataset, scVI outperforms scDGD in terms of RMSE, but achieves again lower clustering performance and requires more computational resources (Supplementary Table S2).

### 4 Discussion

In this work, we have presented the DGD, a Bayesian formulation of a deep generative neural network using MAP estimation for model parameters and representations. This is a fully tractable and simple approach, for which no encoder is needed. This gives the DGD the advantage of having fewer parameters and fewer modelling choices, such as encoder architecture. We have shown that the DGD achieves good reconstruction,
clustering, and generative performance on much smaller latent spaces than some of the other generative approaches, such as VAEs. In this sense, it is comparable to flow-based models (Xiao et al. 2019), but with less architectural restrictions. Additionally, we see clear advantages of this approach due to the simplicity of incorporating more complex distributions for modelling the latent representation. Analogous to our results on Fashion-MNIST, others have reported much better clustering and generative performances with a mixture of Gaussians compared to classic VAEs with standard Gaussians as priors over latent space (Grønbech et al. 2020).

In our application to single-cell expression counts, we showed that we can achieve reconstruction and clustering performances superior to those of comparable models, but with much smaller and better structured latent spaces, fewer parameters, and more detailed distributions. Our choice for comparison are scVI (Gayoso et al. 2022) and scVAE (Grønbech et al. 2020), two VAE-based methods modelling the representation of single-cell expression data. Overall, we compare scVI and our model on a total of 12 datasets, ranging from 500 to over 1 million cells. In these experiments, scDGD largely outperforms scVI on both reconstruction and clustering.

Figure 4. UMAP projection of CD34-positive cell latent representations learned by scDGD with 18 Gaussian components coloured by expression levels of different genes. (a) Heatmap of Spearman correlation coefficients between GMM component probabilities and marker expression counts for all CD34-positive samples. (b) Representations and component means (numbers) in UMAP projection of all CD34-positive cells on black background. (c) UMAP representation projection coloured by expression counts for gene markers indicated by the plot titles.
performance. Investigating one of the learned representations closer, we find that the structure shows meaningful relationships and sub-clustering by Gaussian components, which can be linked to different sub-populations in some cell types.

A caveat of this model is the inference time of new data points. The lack of an encoder means that the representations of new data points need to be found by back-propagation (with fixed model parameters). This process is more time-consuming and computationally expensive than passing new data points through an encoder as in the VAE-based model. However, in our experiments this only took 0.93 s to 36.24 min for 59–130 613 test cells, respectively. In the case where inference is done repeatedly, one could also train an encoder afterwards as described in Schuster and Krogh (2021b), which still results in a more expressive representation than the traditional AE setup.

Altogether, we find that the DGD is a simple and efficient approach to learning low-dimensional representations. Unlike some other generative models, it is fully tractable and bears no architectural restrictions. Although the model can be supervised as we showed on Fashion-MNIST, our scRNA model is completely unsupervised and is suitable for data without existing labels, while achieving a meaningful clustering of the representation on its own. We also observed that a more complex distribution relating to more GMM components can be beneficial for discovering more substructures in the data than provided by class labels. This makes the DGD an ideal candidate for modelling a multitude of biological and medical data. For these types of data, interpretability of the latent space is crucial. Several downstream applications, such as perturbations and pseudo-time, require a low-dimensional and continuous representation. We plan to expand the model to a variety of biological tasks and to include semi-supervised learning and other features that are of interest to generative models for biological and medical data. We further aim to apply this approach to even more extensive single-cell expression datasets and multi-omics data that allow a more thorough investigation of the latent space and can provide meaningful tools for data integration and downstream analysis.

Acknowledgements
We acknowledge all the great discussions on representation learning and generative models with Wouter Boomsma, Jes Frølksen, Søren Hauberg, Aasa Faragå, Ole Winther, and other members of Center for Basic Machine Learning Research in Life Science (MLLS), as well as support from our Center of Health Data Science on discussions about the methods and single-cell data (especially Inigo Prada Luengo) and for reviewing our manuscript (Jennifer Anne Bartell, Inigo Prada Luengo, and Thilde Tørkelsen). We especially thank Sarah Teichmann, her lab and Emma Dann for the insight we have gained into single-cell sequencing data, which was invaluable in the reviewing process.

Author contributions
A.K. contributed to the theoretical foundation of the approach. V.S. contributed to the experiments and figures. A.K. and V.S. contributed to the implementation of the approach and the writing of the manuscript.

Supplementary data
Supplementary data are available at Bioinformatics online.

Conflict of interest
None declared.

Funding
This work was supported by grants from the Novo Nordisk Foundation [NNF20OC0062606, NNF20OC0059939, NNF20OC0063268 to A.K.].

Figure 5. Performance comparison of scVI and scDGD on independent datasets. Error bars indicate the standard error of the mean from three models trained with different random seeds. Asterisks indicate whether differences are significant based on an alpha value of 0.05. (a) Reconstruction performance measured as the NLL of the negative binomial distribution and the RMSE. (b) Clustering performance measured by the ARI.
Data availability
No datasets were generated during this study. All data used in this work are publicly available and referenced in Materials and methods.

Code availability
scDGD is available as a python package at https://github.com/Center-for-Health-Data-Science/scDGD. The remaining code is made available here: https://github.com/Center-for-Health-Data-Science/dgd.

References


