Review of the paper: scVAE, Variational auto-encoders for single-cell gene expression data

Akedjou ADJILEYE & Imane SI SALAH

December 22, 2023

1 Description of the problem

Single-cell RNA sequencing (scRNA-seq) allows the analysis of gene expression at the level of individual cells. This provides the means to investigate the complexities of gene dysregulation in specific cell types and how it contributes to the formation of some diseases. To cluster cells into putative cell types upon the processing of the genes, existing methods often involve computationally intensive processes including normalization, transformation, decomposition, embedding, and clustering of gene expression levels. Despite their results, these methods are computationally expensive. In response to this challenge, the authors of [1] propose an innovative strategy using expressive deep generative models, capable of capturing complex relationships within raw data in depth, thereby by passing the need for extensive preprocessing. In their exploration, the authors specifically turn to Variational Autoencoders (VAEs), a probabilistic model whose performance is quantified through likelihood functions—an invaluable metric for robust model comparison. Additionally, the authors explore the performance of the Gaussian Mixture VAE (GMVAE), demonstrating its superior ability to yield higher-quality clustering compared to traditional methods.

2 Latent variable models

2.1 Varitional AutoEncoder (VAE)

The VAE operates as a set of interconnected and yet independently parameterized models, namely an encoder and a decoder. Given an input data (in general with very high dimension) $\mathbf{x}$, the encoder provides the decoder an estimate of the posterior distribution of some latent random variables which serve as a low dimensional representation of $\mathbf{x}$. The decoder takes that latent representation to regenerate the input data. These two models are usually neural networks that learn by maximizing an evidence lower bound of the input data likelihood. At the end of the training, the encoder can be used to encode data and perform operations like clustering in the lower dimensional latent space, while the decoder can be used to generate new data given some arbitrary latent representations.

In the scope of this project, we assume that our input data $\mathbf{x}$ is drawn from a distribution $p_\theta(\mathbf{x})$ parameterised by θ, where $\mathbf{x} = (x_1, x_2, ..., x_N)$ represents a single cell with N features, and each feature corresponds to a gene expression count. Given that N is very large, a latent variable $\mathbf{z}$ (which serves a code for x) with dimension $<< N$ is introduced. Then the joint probability of $\mathbf{x}$ and $\mathbf{z}$ is:

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z}) \tag{1}$$

where:
$p_\theta(\mathbf{x}|\mathbf{z})$: is the conditional likelihood function
$p_\theta(\mathbf{z})$ is the prior probability of $\mathbf{z}$

Then to obtain the distribution of $\mathbf{x}$, we marginalize out $\mathbf{z}$ as follows:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z} \tag{2}$$

Then introducing the log and summing over all data samples, we obtain the total log-likelihood (under the data independence hypothesis):

$$F(\theta) = \sum_{m=1}^{M} \log p_\theta(\mathbf{x}_m) \tag{3}$$

with M being the number of cells.
Our objective for what remains is to find a θ that maximizes the log-likekihood fucntion.

Given that the features represent the numbers of gene expression count, we model each feature x_k of $\mathbf{x}$ with a Poisson distribution with a parameter λ_k:

$$f(x_k; \lambda_k) = e^{-\lambda_k} \frac{\lambda_k^{x_k}}{x_k!}$$

and the conditional likelihood becomes:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{N} f(x_k; \lambda_k) \tag{4a}$$

$$p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I}) \tag{4b}$$

We choose the prior $p_\theta(\mathbf{z})$ to be Gaussian and the parameters of the Poisson distribution $\lambda = (\lambda_1, .., \lambda_N)$ that depend on $\mathbf{z}$ will be predicted using a single layer feedforward network parameterized by $\theta = [\mathbf{W}, \mathbf{b}]$, as follows:

$$\lambda_\theta(\mathbf{z}) = h(\mathbf{W}\mathbf{z} + \mathbf{b}) \tag{5}$$

with $\mathbf{W}$ and $\mathbf{b}$ be the weights and biases of the network, h its activation function (to ensure positivity of λ). Note that we can also use a multi-layer perceptron (MLP) to predict $\lambda_\theta(\mathbf{z})$.

That represents the **decoding (or generation) part** of the VAE.

Moving on to the **encoding part**, we aim to predict the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ of $\mathbf{z}$. Using the Bayes rule, we have:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{z})} = \frac{p_\theta(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z})}$$

Therefore:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}$$

Since the marginal likelihood is intractable (product of a Poisson and a gaussian), we'll approximate the $\mathbf{z}$ posterior $p_\theta(\mathbf{z}|\mathbf{x})$ by a tractable distribution $q_\phi(\mathbf{z}|\mathbf{x})$ depending on a parameter ϕ, by minimizing the following Kullback-Leibler (KL) divergence:

$$\begin{aligned}
KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})] &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\
&= \int \log\left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right) q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\
&= \int \log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}\right] q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}\right] d\mathbf{z} + \log p_\theta(\mathbf{x}) \int q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z}
\end{aligned}$$

Since $\int q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} = 1$ as q is a probability density and by setting

$$\mathcal{L} = \int q_\phi(\mathbf{z}|\mathbf{x}) \log\left[\frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] d\mathbf{z},$$

we get

$$\log p_\theta(\mathbf{x}) = KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})] + \mathcal{L} \tag{6}$$

Since $KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})] \geq 0$, we have $\log p_\theta(\mathbf{x}) \geq \mathcal{L}$, so $\mathcal{L}$ is a lower bound of the total log-likelihood of $\mathbf{x}$ and is called the evidence lower bound (ELBO). We'll then maximize the ELBO to get it as closest as possible to $\log p_\theta(\mathbf{x})$.

We further develop the expression of $\mathcal{L}$ as follows:

$$\mathcal{L} = \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) - \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - KL\left[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right] \tag{7}$$

This last expression shows that maximizing the ELBO account for minimizing $KL\left[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right]$ from one side, ensuring our control on the divergence between the prior of $\mathbf{z}$ and its porterior over $\mathbf{x}$ (**the regularization term**), and from other side maximizing the expectency of the likelihood over the posterior distribution (**the good prediction term**).

In practice, $q_\phi(\mathbf{z}|\mathbf{x})$ will be chosen to be gaussian $\mathcal{N}(z; \mu_\phi(\mathbf{x}), \sigma^2_\phi(\mathbf{x}))$ and the encoder will predict its parameters $\mu_\phi(\mathbf{x})$ and $\sigma^2_\phi(\mathbf{x})$.

2.2 Gaussian Mixture Variational AutoEncoder (GMVAE)

Using a gaussian distribution as the prior probability distribution of $\mathbf{z}$ only allows for one mode in the latent representation. If there is an inherent clustering in the data (imagine the cells represent different cell types), it is desirable to have multiple modes, e.g., one for every cluster or class. This can be implemented by using a gaussian-mixture model in place of the gaussian distribution.

To do this, we introduce a categorical latent variable $\mathbf{y}$ to the VAE model. The joint probability of $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ is then:

$$p_\theta(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}|\mathbf{y}) p_\theta(\mathbf{y})$$

where:
$p_\theta(\mathbf{x}|\mathbf{z})$: is the the same Poisson distribution as for VAE
$p_\theta(\mathbf{z}|\mathbf{y}) = \mathcal{N}(\mathbf{z}; \mu_\theta(\mathbf{y}), \sigma^2_\theta(\mathbf{y})\mathbf{I})$
$p_\theta(\mathbf{y}) = \mathcal{M}(\mathbf{y}; 1, \pi)$, a multinomial distribution with paramater $n = 1$ and $\pi = (\pi_1, ..., \pi_K)$, the probability vector of size K, the number of components in the gaussian mixture. π is supposed to be uniform with no prior information one the data.

2.2.1 Encoding process

Given an input data x, a multi-layer perceptron (MLP) will be used to first predict the categorical variable y and the posterior probability parameters of $\mathbf{y}|\mathbf{x}$ will be estimate using the classes' probabilities predicted. Then given $\mathbf{y}$ and $\mathbf{x}$, a MLP will also be used to predict the approximate mixture of gaussian posterior parameters $\mu_\phi(\mathbf{x}, \mathbf{y})$ and $\sigma^2_\phi(\mathbf{x}, \mathbf{y})$.

2.2.2 Decoding process

The decoding process is essentially the same as for VAE except that now, the latent variable $\mathbf{z}$ that was used to feed a MLP to predict the Poisson distribution λ is also dependent of the categorical variable $\mathbf{y}$.

2.2.3 ELBO of a GMVAE

The lower bound of a GMVAE has the same format as for VAE in (7) with $\mathbf{z}$ replaced by $\mathbf{z}|\mathbf{y}$ and an additional regularization term which controls the KL divergence between the prior distribution of $\mathbf{y}$ and its posterior. We have:

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{y}|\mathbf{x})} \left[\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - KL\left[q_\phi(\mathbf{z}|\mathbf{y})||p_\theta(\mathbf{z}|\mathbf{y})\right]\right] - KL\left[q_\phi(\mathbf{y}|\mathbf{x})||p_\theta(\mathbf{y})\right] \tag{8}$$

In the first term, we recognize in the expectation as being the same ELBO as in (7), applied to $\mathbf{z}|\mathbf{y}$. This expectation is the **good prediction term** of a GMVAE as it ensures that the ELBO of the posterior distribution of $\mathbf{z}|\mathbf{y}$ is high under the posterior distribution of $\mathbf{y}$ and the second term is the **regularization term** which control $\mathbf{y}$.

3 Experiments

3.1 Practical Hypothesis

In this part, we will describe how we implement a variational autoencoder (VAE) and a gaussian mixture variational autoencoder (GMVAE) from scratch using Pytorch.

3.1.1 Optimization problem

As said in section 2.1, we define an approximating variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ parametrized by ϕ and minimize its Kullback-Leibler (KL) divergence with the unknown true posterior $p_\theta(\mathbf{z}|\mathbf{x})$. This is equivalent to maximizing the evidence lower bound (ELBO) (7) with respect to $q_\phi(\mathbf{z}|\mathbf{x})$.

Let's first rewrite the KL-divergence term in (7):

$$KL\left[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}(\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{z})\Big).$$

As for the likelihood term in (7), the KL divergence can be written as an expectation over the approximate distribution $q_\phi(\mathbf{z}|\mathbf{x})$.

By assuming that samples are i.i.d, maximizing the ELBO is equivalent to minimizing the following loss:

$$-\sum_{m=1}^{M} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big(\log p_\theta(\mathbf{x}_m|\mathbf{z}) - \frac{1}{M}(\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{z}))\Big)$$

The KL-divergence is not dependent on the generation parameter λ , so we compute directly after the sampling in the inference process (fig 1).

3.1.2 Monte Carlo estimator

Deriving those expectations can be some tedious mathematics, or maybe not even possible. Luckily we can get estimates of the mean by taking samples from $q_\phi(\mathbf{z})$ and average over those results.

Even more simple, using only one sample is still an unbiased gradient estimator. Hence, loss function simply boils down to minimzing at each step:

$$\mathcal{L}_{MC} = -\sum_{m=1}^{M}\Big(\log p_\theta(\mathbf{x}_m|\mathbf{z_s}) - \frac{1}{M}(\log q_\phi(\mathbf{z}_s) - \log p_\theta(\mathbf{z}_s))\Big) \tag{9}$$

where $\mathbf{z}_s \sim q_\phi$ is a sample from the approximate distribution, MC is for Monte-Carlo.

3.1.3 Reparametrization trick

If we start taking samples from q_ϕ, we leave the deterministic world, and the gradient can not flow through the model anymore. We avoid this problem by reparameterizing the samples $z_s \sim \mathcal{N}(\mu, \Sigma^2)$ from the distribution.

Instead of sampling directly from the approximate distribution, we sample from a centered isotropic multivariate gaussian and recreate samples from the variational distribution. Now the stochasticity of $\boldsymbol{\varepsilon}$ is external and will not prevent the flow of gradients.

$$\mathbf{z}_s = \mu + \Sigma \odot \boldsymbol{\varepsilon}_s$$

where $\boldsymbol{\varepsilon}_s \sim \mathcal{N}(0, \boldsymbol{I})$. For more simplicity, we also assume that all the components of $\mathbf{z}_s$ have the same variance σ^2 that we compute using the softplus function $\log(1 + e^\rho)$ with a learnable parameter ρ to ensure the positivity of σ^2. Also, to avoid numerical overflow, we forced ρ to be smaller than 100, as e^ρ can quickly return infinity do to limited bit precision.

3.1.4 Illustration of the inference (encoding) and generation (decoding) process

We code our models following the described process in the paper [1].

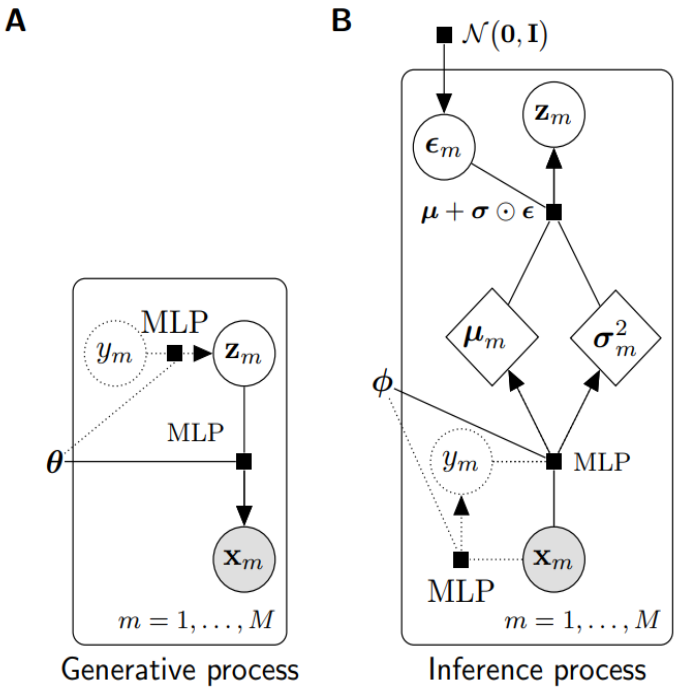

Figure 1: Model graphs of (A) the generative process and (B) the inference process of the variational auto-encoder models. Dotted strokes designate nodes and edges for the Gaussian-mixture model only. Grey circles signify observable variables, white circles represent latent variables, and rhombi symbolise deterministic variables. The black squares denote the functions next to them with the variables connected by lines as input and the variable connected by an arrow as output; image from the paper (see [1])

3.2 VAE on synthetic data

To ensure the effectiveness of our VAE model, we generated a synthetic dataset comprising 10 features and 2000 samples. These samples were drawn from a Poisson distribution with parameters $\lambda = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$. Our goal was to reconstruct the parameter λ of the dataset distributions. To infer the parameters μ and σ^2 of the latent variables, we utilized an MLP with one hidden layer with 7 neurons for both the encoder and the decoder. The input was encoded into a 3 dimensional latent variable $\mathbf{z}$.

During training, we minimized the loss $\mathcal{L}_{MC}$ in Equation (9) using the Adam Optimizer with a constant learning rate of 0.1 and a weight decay of 5e-2. The model was trained for 150 epochs, and we visualized the ELBO, log-likelihood, and KL divergence in Figure 2.

As anticipated, the log-likelihood increased over epochs, and the KL divergence decreased. This ensures a good reconstruction of the Poisson input data over the posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and a proper balance between the prior and posterior of $\mathbf{z}$. Additionally, the barplot of the reconstructed parameter λ indicates that the model effectively learned the distribution of the input data.

3.3 VAE on real gene expression data

In this project we consider a subset of the scRNA seq for mouse brain cell that contains gene counts for $20,000$ cells (MB-20K) with 27 998 features (see Table 1) splitted into train and test sets with proportion 81% train and 19% test.

Data set	Examples	Features	Sparsity [%]
Mouse brain cells (20k)	20 000	27 998	92.82

Table 1: Mouse brain cells (20k) Dataset

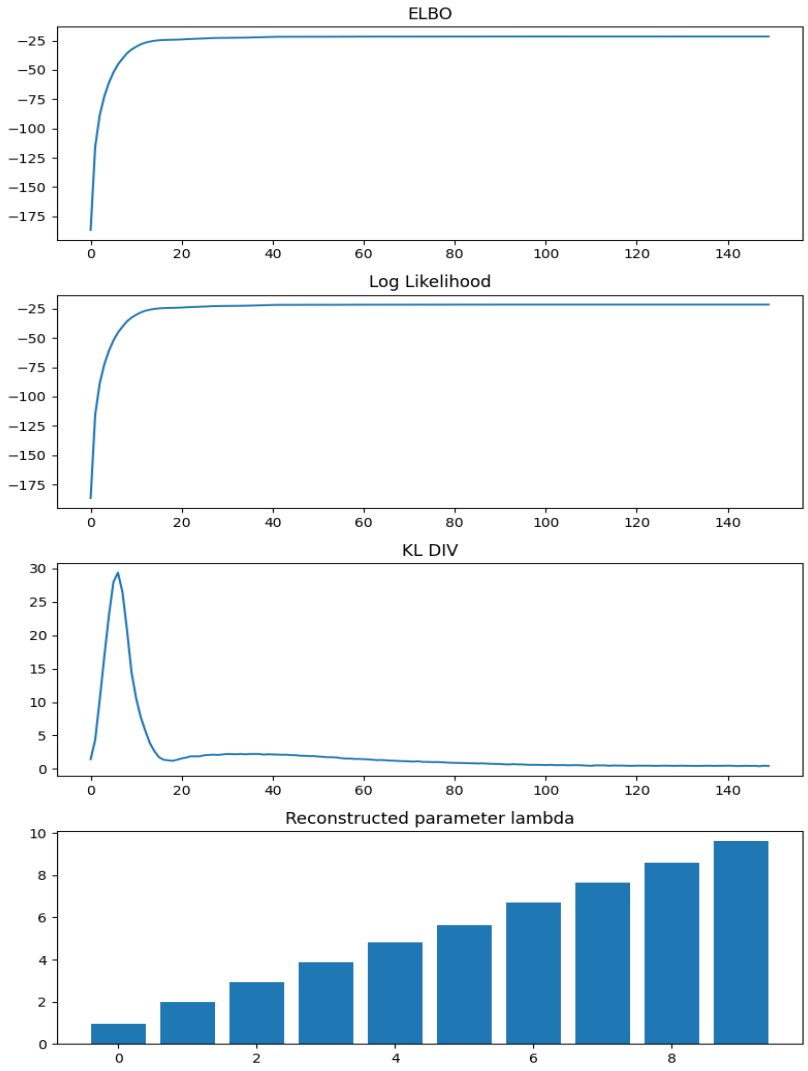

Figure 2: VAE result on synthetic Poisson data

We conducted a grid search to determine the best model parameters by training the network for one epoch with multiple hidden layers, varying the number of units between 100, 250, 500, and 1000. Additionally, we varied the latent dimension of $\mathbf{z}$ between 0 and 100, as reported in the article. (see our code for more details about the gird search).

This search was performed for the following reconstruction distributions: Poisson, Negative Binomial (NB), Zero-Inflated Poisson (ZIP), and Zero-Inflated Negative Binomial (ZINB), and the results were recorded.

Through this search, we identified the best combination of parameters that resulted in the maximum ELBO value. The results are displayed in Table 2.

The Negative Binomial (NB) distribution is also a suitable distribution proposed in [1] to model and capture the variability of count data; it have two parameters, the mean λ and the dispersion parameter r, which will be learned by the decoder in practice. Its density is expressed as:

$$f(x_k; \lambda_k, r_k) = \binom{x_k + r_k - 1}{x_k} p_k^r (1-p)_k^x$$

Additionally, we also explored the proposed Zero-Inflated Negative Binomial (ZINB) distribution proposed in [1] as an alternative as its accommodates excess zeros in the count data and introduces a separate process for zero generation. It extends the conventional NB distribution by including a point mass at zero, providing a more robust framework for modeling scRNA-seq data, which often exhibits

Reconstruction distributions	Encoder Hidden Size	Decoder Hidden Size	Latent Dimensio	ELBO
Poisson	[100, 250, 500, 1000]	[100, 250]	50	-12743.27
Negative Binomial	[100]	[100, 250, 500, 1000]	100	-6992.24
Zero Inflated Poisson	[100]	[100, 250, 500, 1000]	50	-12492.06
Zero Inflated Negative Binomial	[100]	[100]	100	-10834.93

Table 2: Grid search results: the entries in the columns for encoder and decoder hidden sizes are the optimal architectures for each distribution. The length of each entry denotes the number of layers in the network, while the content of the entry indicates the number of hidden units in each layer. For instance, a decoder hidden size of [100, 250] implies a two-layer decoder, with the first layer having 100 hidden units and the second layer having 250 hidden units. ELBO denotes the optimal loss obtained; note that each training in the grid search lasted only one epoch as **of limited computational resources**

excessive zero counts. The density function of the ZINB distribution incorporates the zero-inflation parameter π_k:

$$f(x_k; \lambda_k, r_k, \pi_k) = \begin{cases} \pi_k + (1-\pi_k) \cdot \binom{x_k + r_k - 1}{x_k} p_k^{r_k} (1-p_k)^{x_k}, & \text{if } x_k = 0, \\ (1-\pi_k) \cdot \binom{x_k + r_k - 1}{x_k} p_k^{r_k} (1-p_k)^{x_k}, & \text{if } x_k > 0. \end{cases}$$

We trained the VAE model for 500 epochs for each of the mentioned distributions, using the corresponding parameter configurations from the grid search. The objective of the training was to minimize the loss $\mathcal{L}$, as mentioned in the previous section, using the Adam Optimizer with a fixed learning rate of 1e-4 and a weight decay of 5e-2. Additionally, we incorporated batch normalization layers to make the model less sensitive to weight initialization and to help stabilize the optimization process. The recorded results are shown in Table 3.

Reconstruction distribution	ELBO
Poisson	-7178.45
Negative Binomial	-6568.23
Zero Inflated Poisson	-7176.42
Zero Inflated Negative Binomial	-7181.14

Table 3: Testing results

From Table 3 we see that the NB reconstruction distribution resulted in the highest ELBO value followed by ZIP which is not quite in accordance with the results shown in the paper. For the Poisson, training the model with the optimal configuration returned from the grid search led to a lot of numerical instability (see 3.3.1). We noticed that the parameter configuration used for training has a deeper encoding network with 4 layers compared to other distributions, this maybe increase the risk of having exploding or on the contrary vanishing gradients; to obtained the ELBO value in table 3, we used only one layer with 100 units in the encoder and kept the rest of the combination unchanged, which led to better results.

3.3.1 Numerical issues with the Poisson distribution

Given a sample $\mathbf{x} = (x_1, ..., x_N)$ of the MB-20k dataset with N the number of features, we want to reconstruct his supposed Poisson distribution parameter $\lambda = (\lambda_1, ..., \lambda_N)$. The likelihood of $\mathbf{x}$ is

$$f(\lambda, \mathbf{x}) = \prod_{k=1}^{N} e^{-\lambda_k} \frac{\lambda_k^{x_k}}{x_k!} = \prod_{k=1, x_k=0}^{N} e^{-\lambda_k} \prod_{k=1, x_k>0}^{N} e^{-\lambda_k} \frac{\lambda_k^{x_k}}{x_k!}$$

. Let denote by

$$A = \prod_{k=1, x_k=0}^{N} e^{-\lambda_k} = e^{-\sum_{k=1, x_k=0}^{N} \lambda_k}$$

the first term in $f(\lambda, \mathbf{x})$, and $N_0 = card\{\lambda_k | x_k = 0, k = 1, ..., N\}$.

In practice, the MB-20k dataset has a very high sparsity (> 0.92) and we have $N_0 = 25987$ in average, which lead to very high value for the quantity $B = \sum_{k=1,x_k=0}^{N} \lambda_k$ such as his numerical exponential can returns 0.0 because of limited bit precision. That given, the loglikelihood term in the ELBO will return -infinity, which is problematic for the optimization algorithm and for the training.

We compute B at the end of the forward of the VAE for one training attempt with the Poisson reconstruction; the training last 38 epochs before we get numerical issue with a loss equal to -inf. Figure 3 shows the progression of B.

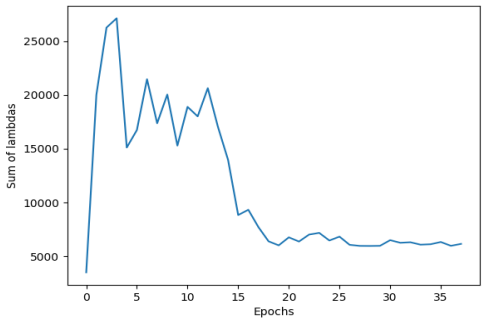

Figure 3: B values during training with the Poisson reconstruction

It's can be curious to see that B have reached a very high value greater than 25000 without any numerical issue in the loss. For instance, e^{-1000} in Pytorch returns 0.0. But that's normal because in the implementation of the formula for A, each $e^{-\lambda_k}$ is calculated separately in the Poisson loglikelihood function in Pytorch and are then multiplied. Therefore, it would be interesting to look at the highest value $\lambda_{k_{max}}$ of $(\lambda_k)_k$ during the training. Figure 4 shows that values during another training attempt. The maximum values reached is around 400; in that case it's smaller than the limited value for which the numerical exponential returns 0.0.

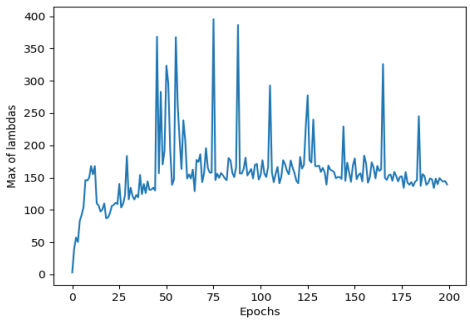

Figure 4: $\lambda_{k_{max}}$ values during training with the Poisson reconstruction

This numerical instability with the Poisson loglikelihood motivates the use a zero-inflated Poisson (ZIP) distribution, that was already introduced in [1] but without any real justification. The authors just claims that the ZIP controls the amount of excessive zeros added to an existing probability distribution. We see our result as a more convinced justification with theorical and practical proof.

The ZIP density function for $x, \lambda \in \mathbb{R}, \mathbb{R}_+$ is computed as follows:

$$g_{zip}(x; \rho, \lambda) = \begin{cases} \rho + (1-\rho)g(x;\lambda), & \text{if } x = 0, \\ (1-\rho)g(x;\lambda), & \text{if } x > 0, \end{cases}$$

where ρ represents here the probability of excessive zeros (can be estimated by the dataset sparsity)

and $g(x;\lambda) = e^{-\lambda}\frac{\lambda^x}{x}$. It's easy to see that g_{zip} is still a probability density as

$$\sum_{x\in\mathbb{N}} g_{zip}(x;\rho,\lambda) = \rho + (1-\rho)\sum_{x\in\mathbb{N}} g(x;\lambda) = \rho + 1 - \rho = 1$$

as $\sum_{x\in\mathbb{N}} g(x;\lambda) = 1$ because g is the Poisson density function.

The idea of ZIP is to avoid the vanishing of the factor A in the likelihood $f(\lambda, \mathbf{x})$ by having a strictly positive constant control term, which depends on the sparsity of the dataset. Now, even if $\lambda_{k_{max}}$ is too high as his numerical exponential returns 0.0, there still the term ρ that ensures the non-vanishing of the likelihood and by the way the non-exploding of the loglikelihood term in the ELBO loss.

3.4 GMVAE

3.4.1 Implementation details

We followed the architecture illustrated in figure 1 (B) to implement the gaussian mixture variational autoencoder. In the forward process, we add the categorical variable y_m prediction block to the inference code of the simple VAE, and to feed the MLP that predict the latent variable parameters, we use a concatenation of the VAE input vector $\mathbf{x}_m$ and a one-hot encoded of the predicted category y_m for $\mathbf{x}_m$.

One **ambiguous** point in the generative process described in the paper (fig 1, **A**) is about the black square that connect y_m to $\mathbf{z}_m$ with an MLP parameterized by $\theta \neq \phi$. It's seems like we should predict another latent variable $\mathbf{z}_m$, but we didn't think that would make sense and as in the VAE, we directly use the infered latent variable $\mathbf{z}_m$ to feed the decoder MLP and reconstruct $\mathbf{x}_m$.

For ELBO loss for GMVAE in formula (8), we use the same Monte-Carlo estimator as in 3.1.2 for VAE and minimized

$$\mathcal{L}_{MC} = -\Big(\sum_{m=1}^{M}\Big(\log p_\theta(\mathbf{x}_m|\mathbf{z_s},\mathbf{y}) - \frac{1}{M}[\log q_\phi(\mathbf{z}_s|\mathbf{x}_m) - \log p_\theta(\mathbf{z}_s)]\Big) - \frac{1}{M}[\log q_\phi(y_m) - \log p_\theta(y_m)]\Big), \tag{10}$$

where $\mathbf{z}_s \sim q_\phi$ is a sample from the approximate distribution of $\mathbf{z}|\mathbf{y}$, MC is for Monte-Carlo.

There is an **important** point in this formula. Because what we do is to directly predict the category of each sample $\mathbf{x}_m$ using a MLP, there is no real generation process for $\mathbf{y}$, we use the predicted classes's probabilities as parameters for the categorical posterior of $\mathbf{y}$ and a uniform probability vector for the prior parameter. That given, the KL divergence term on $\mathbf{y}$ can easily be replaced by a **cross-entropy loss** as it's really look likes a classical classification task.

3.4.2 Experimentation on synthetic dataset

We used the same strategy as in 3.2 for VAE to create a synthetic dataset with 10 features and 10k samples to test our implementation of the GMVAE. For categories, we runned a KMeans clustering with **5** clusters and use the predicted clusters as ground thruth labels. We use the same training method and the same model architecture as in 3.2, the MLP that predicts $\mathbf{y}$ have one hidden layer with 10 neurons and a ReLU actiavtion; we used a softmax actiavtion in the output layer. Sometimes, we got numerical **issue** with the loss in (10) due to the Poisson reconstruction loglikelihood used for that experiment.

Ablation study on the regularization term in the ELBO: First, we remove the categorical variable $\mathbf{y}$ regularization term in (10) and train the model with only the good prediction term. Secondly, we replace the KL divergence regularization term with a cross-entropy loss between the predicted classes and the ground truth categories. Results are presented in figure 5. The model learns well to reconstruct the Poisson distribution but we still have to improve to better handle the categorical variable $\mathbf{y}$.

3.4.3 Other Problems encoutered

It was very complicated to train the GMVAE, even on the simple synthetic dataset. In one part, the loss backward process in Pytorch gave many errors, with the most often been this runtime error

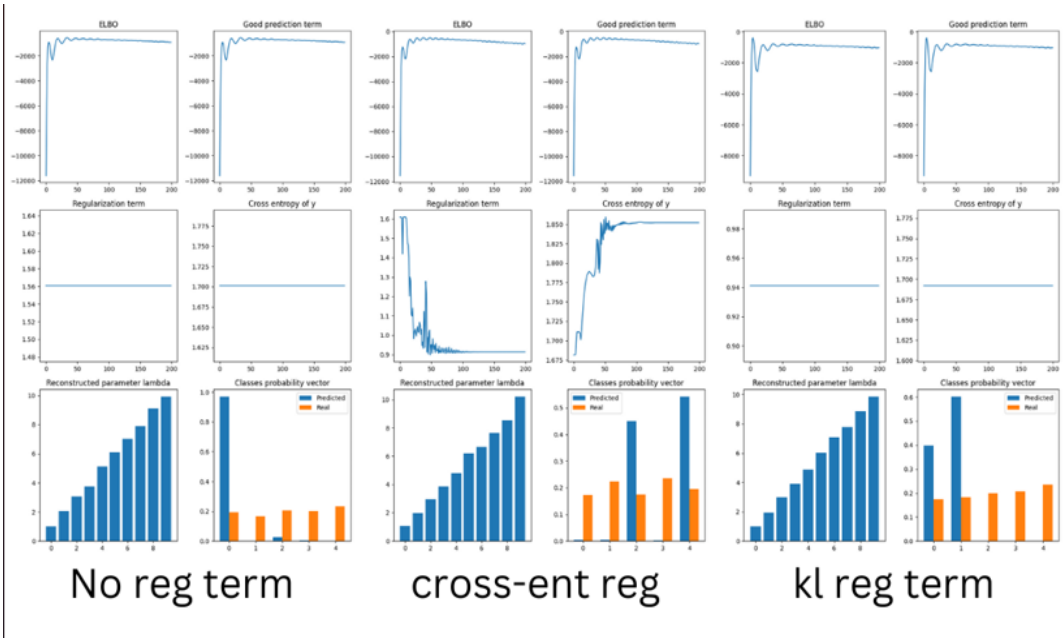

Figure 5: Ablation study on the loss of GMVAE: The Poisson parameter is well reconstructed in all case; the cross entropy regularization seems better handle the categorical parameters learning than the other, but the predicted categorical probabilities are bad in all cases

.

message: *"Trying to backward through the graph a second time, but the buffers have already been freed. Specify retain_graph=True when called backward for the first time"*. We spent a lot of time trying to debbug that without success. So we were forced to put the *retain_graph* parameter to *True* during training, which significantly slows it down. In the other part, there are some ambiguous points in the paper about the generation process, we got a lot of numerical issue relative to the reconstruction distribution, the categories predictions is still very bad and we spent a lot of times trying to improve it without real success.

4 Discussion and Conclusion

In this project, we implemented both a Variational Autoencoder (VAE) and a Gaussian Mixture VAE from scratch, drawing inspiration from the architectures presented in [1]. To achieve this, we explained how the inference and generation processes work at the level of the encoder and decoder, respectively, demonstrating how optimizing the model is equivalent to maximizing the Evidence Lower Bound.

We conducted experiments to test our model on a synthetic dataset we created, and observed the convergence of the ELBO, log-likelihoods, and KL divergences. Subsequently, we performed experiments on real scRNA sequences using different likelihood distributions. After selecting the best model architecture through a grid search, we transitioned to the GMVAE. Here, our focus shifted to training it on another synthetic dataset containing multiple classes.

Initially, our goal was to dive deeper into clustering within the latent space, comparing the VAE and GMVAE while benchmarking our results against those reported in [1]. However, we encountered a lot of challenges. The first one is the unavailability of labels for the scRNA datasets used in the paper. Additionally, during training, we realized that while our theoretical aim was to maximize the ELBO, the Adam optimization algorithm implemented in PyTorch functions as a minimizer. Therefore, we needed to adjust the loss function to return the negative of the ELBO. Moreover, adding more layers to the encoder and decoder to create deeper neural networks brought additional challenges, particularly in dealing with vanishing and exploding gradients. At the end we were able to run the VAE on a synthetic dataset at first then noting that the results were quite satisfying, we run it on the real scRNA sequence dataset described in Table 1, and the results on the test set were as displayed in Table 3. as for the GMVAE, after a long trial and error and debugging process we were able to run it

on a synthetic labelled dataset that we created although the clustering part did not work.

5 Unexplored aspects

- We would like to present some results on the vizualisation of the latent space of real data and performs clustering, but the experimentation was too complicated and we didn't have enough time.
- We would like to compare the clustering on the latent space of the VAE and GMVAE on a synthetic dataset, but we spent too much time just trying to make the GMVAE code work. Maybe we'll add this point on the poster.

6 Specific contributions

- **Akedjou ADJILEYE**: GMVAE implementation, experimentation on synthetic datasets with VAE and GMVAE, Poisson distribution instability investigation.
- **Imane SI SALAH**: VAE implementation, ZIP, NB and ZINB distributions investigation, experimentation on real MB-20k data, including grid search and full training with all 4 distributions.

7 Reference

1 scVAE: Variational auto-encoders for single-cell gene expression data; *Christopher H Grønbech, Maximillian F Vording, Pascal N Timshel, Capser K Sønderby, Tune H Pers and Ole Winther*