Unsupervised neural methods for modelling cell differentiation

Ioana Bica
Churchill College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Department of Computer Science and Technology
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: ib354@cam.ac.uk

August 23, 2018

Declaration

I Ioana Bica of Churchill College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14,691

Signed:

Date:

Acknowledgements

Firstly, I would like to thank my supervisor **Dr Pietro Lio'** for his guidance and invaluable advice offered throughout this project. Moreover, I would like to thank **Helena Andres** for brainstorming ideas for the project with me and for offering me help whenever needed.

I would also like to thank **Dr Ana Cvejic** and **Emmanouil Athanasiadis** from the Wellcome Trust Sanger Institute for providing me with the data, as well as for patiently explaining the relevant biological questions that could be answered from it.

I am also thankful to my friends **Michael Painter** and **Malavika Nair** for proofreading this dissertation and for offering me advice to make improvements. Finally, I am also grateful to my family for their unwavering support.

Unsupervised neural methods for modelling cell differentiation

Abstract

Using machine learning techniques to build representations from biomedical data can offer us an understanding of the latent biological mechanism of action and lead to important discoveries. Recent developments in single-cell RNA-sequencing protocols have made possible measuring gene expression for individual cells in a population. Gene expression is the process of synthesising proteins, dictating cell function, based on genetic information. The availability of such data has opened up the possibility of finding answers to biomedical questions about cell differentiation.

In this project, I develop unsupervised neural architectures that can model cell differentiation by building appropriate representations from the high dimensional and complex gene expression data. The unsupervised methods proposed take advantage of the probabilistic framework of the variational autoencoder, capable of modelling the stochastic behaviour of gene expression across the differentiation of cells. Moreover, I propose a computational framework that can be used to analyse the data representations obtained in order to gain more understanding about cell differentiation. Finally, I perform a comparative evaluation of the models constructed in this project and other commonly used dimensionality reduction techniques.

This project builds an appropriate variational autoencoder architecture to model cell differentiation using gene expression data. It represents the first work to apply disentanglement methods based on information theory to improve the latent representation built by the variational autoencoder, and achieve better separation of the biological factors of variation in the gene expression data. It is also the first work to propose an architecture for a graph variational autoencoder, performing spectral graph convolutions, that is suitable for predicting links between cells in a dataset.

The developed computational framework consists of methods for identifying latent representations encoding the differentiation of cells in a dataset, finding driver genes for the differentiation process, changing cellular states through computations on the latent dimension and predicting links between cells. The methodology developed in this sense was successfully applied on a biological dataset consisting of gene expression information on haematopoietic stem and progenitor cells (HSPCs) as well as four types of mature blood cells. The results obtained on this dataset prove the capabilities of the models in answering important biomedical questions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As technology continues to drive biomedical research forward, new challenges arise with the surge of high volume, information-dense and multivariate data that are generated. The extraction of critical information from such data remains an open problem in biomedical research, which can be significantly aided by the incorporation of various machine learning techniques. In particular, unsupervised learning methods have the potential to uncover the underlying structure in biomedical data and therefore propel research on biological processes and diseases that have not yet been fully understood.

This project aims to build unsupervised neural methods that can be applied to understand cell differentiation using gene expression data. Cell differentiation is the biological process through which less specialised cells become more specialised, while gene expression is the process of synthesising proteins, dictating the cell function, based on genetic information. Recent technology for performing single-cell RNA sequencing has resulted in high-throughput experiments capable of measuring gene expression levels for individual cells in a population, thus achieving a granularity not previously possible. In-depth analysis of this high-dimensional and complex gene expression data about the cells can lead to important biomedical discoveries about the factors influencing the differentiation process.

The work done in this project develops a novel methodology relying on unsupervised machine learning techniques that can be leveraged to answer relevant biomedical questions about cell differentiation. Moreover, building an appropriate low dimensional representation of the gene expression data is crucial for subsequent classification or clustering tasks. The unsupervised neural models proposed in this project represent an improvement to the models currently used for performing di-

mensionality reduction on this data.

The methods proposed have broad applicability and can be used on biomedical datasets consisting of various cell types. As a case study, in this project, we will investigate in detail the differentiation of haematopoietic stem cells, i.e. the cells which produce blood cells, by leveraging the computational framework developed in this project.

Unsupervised learning is a type of machine learning capable of finding patterns in a dataset without requiring labelled data points, as it is the case for supervised learning. In the context of biomedical research, unsupervised techniques can reveal the otherwise hidden factors that have caused a particular biological process. To achieve this, unsupervised models build representations from the data which, if thoroughly analysed, can expose latent biological mechanisms of action.

Neural methods, also known as deep learning or neural network models, build such representations by applying non-linear transformations to the input data. These non-linear transformations are performed in a series of layers, such that the first layers extract simple features from the data, while the later (deeper) layers combine these features in more complex and informative representations.

Neural models have already demonstrated an exceptional ability to handle data with non-linear relationships, thus achieving remarkable results on pattern recognition tasks [1–3]. Autoencoders are an application of neural network models to unsupervised learning and they have already been successful in performing non-linear dimensionality reduction, and in embedding the input to a well-known probability distribution [4, 5].

The basis for the models proposed in this project is the variational autoencoder, a type of autoencoder which builds a probabilistic framework for analysing the data. In this framework, we will be interested in approximating the posterior distribution of the latent explanatory factors for the observed experimental data. We will also explore methods for disentangling these underlying explanatory factors in the latent representation. This work represents the first application, to the best of my knowledge, of disentanglement methods for variational autoencoders with the purpose of gaining more understanding from gene expression data.

Additionally, this project develops an unsupervised neural model for predicting relationships between cells by leveraging graph convolutional networks [6, 7] and the variational autoencoder. To the best of my knowledge, this also represents the first application of spectral graph convolutions, as part of a variational autoencoder, for studying gene expression data.

Contributions: The main contributions of this project are as follows:

1. Development of variational autoencoder architecture for modelling the biological processes that influence cell differentiation: **DiffVAE** model.

2. First work to use disentanglement methods to improve the latent representation built by the variational autoencoder from gene expression data. The resulting **Disentangled-DiffVAE** model maximises the mutual information between the input and the latent representation, thus achieving better disentanglement of the biological factors of variation in the gene expression data.

3. First application of Graph Convolutional Networks to gene expression data. The proposed **Graph-DiffVAE** model is a variational autoencoder performing spectral graph convolutions in order to predict links between cells.

4. Computational framework for studying the latent representation built by DiffVAE, Disentangled-DiffVAE and Graph-DiffVAE which includes methods for:

    - identifying the latent dimensions representing to the differentiation of each type of cells;

    - identifying the genes influencing the biological process encoded in the latent representation;

    - changing a less specialised cell into a more specialised one by modifying its latent representation;

    - using gene expression data and an initial input graph for the cells to predict similar links between cells.

5. Comparative evaluation of DiffVAE and Disentangled-DiffVAE with dimensionality reduction techniques such as a simple autoencoder and Principal Component Analysis.

Figure 1.1 gives an overview of the unsupervised neural models and of the methodology developed in this project in order to model cell differentiation. Moreover, the figure highlights the key findings of this project obtained by applying the computational framework to a biological dataset consisting of gene expression measurements from blood (haematopoietic) stem and mature cells.

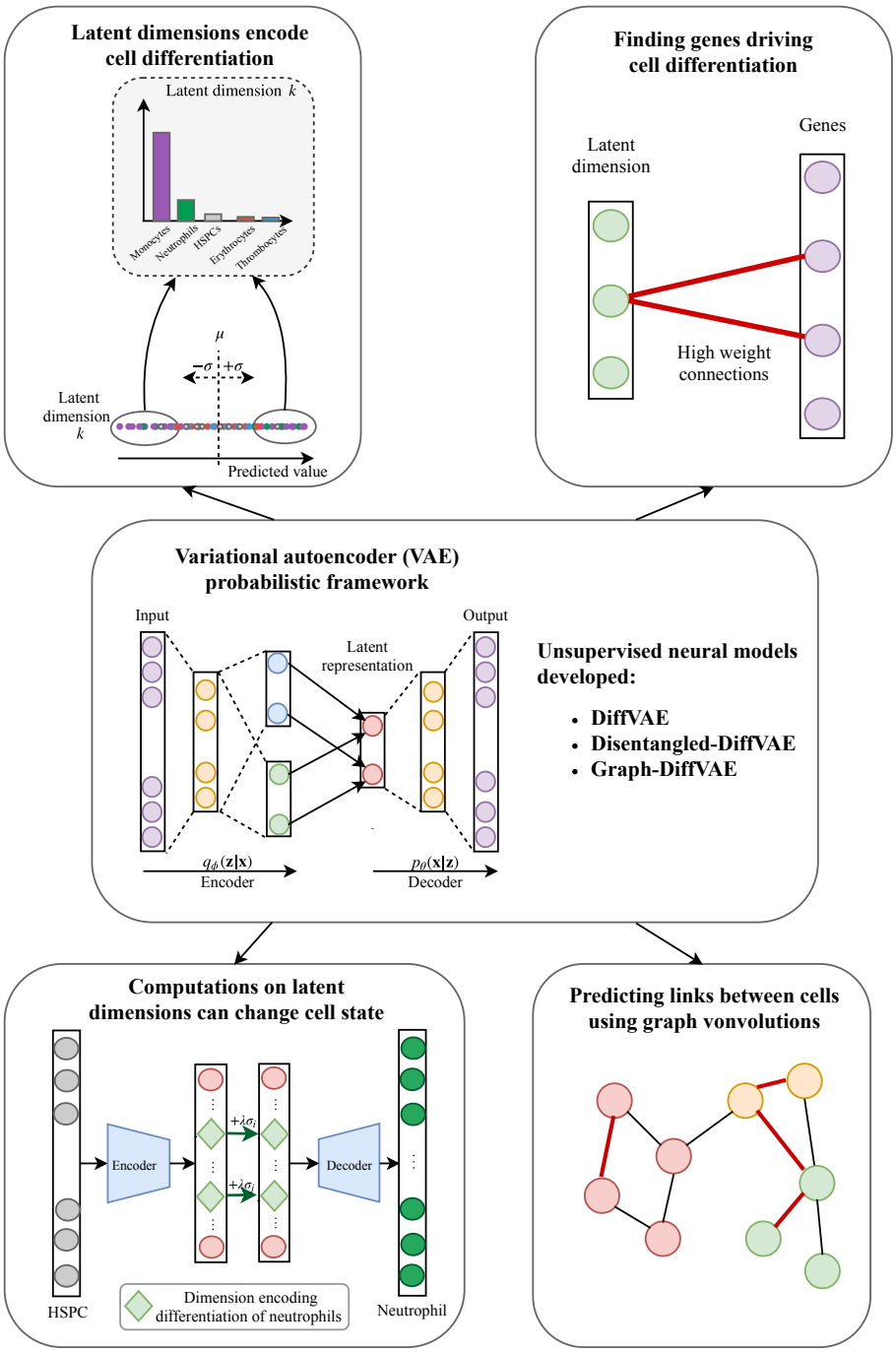


Figure 1.1: Overview of the unsupervised neural models and methodology proposed to model cell differentiation. The neural models developed (described in Chapter 4) leverage the probabilistic framework of the variational autoencoder. The methodology built for understanding the latent data representations (outlined in Chapter 5) leads to important findings on the biological dataset analysed.

The structure of this dissertation is outlined as follows:

- Chapter 2 introduces neural networks, autoencoders and gives a brief overview of RNA-sequencing and of hematopoietic stem cell differentiation.
- Chapter 3 discusses related work both in terms of unsupervised learning techniques and in terms of previous methods applied to gene expression data.
- Chapter 4 presents the design and implementation of DiffVAE, Disentangled-DiffVAE and Graph-DiffVAE.
- Chapter 5 discusses the implementation of the proposed methodology for modelling cell differentiation as well as its evaluation on analysing the differentiation of haematopoietic stem cells.
- Chapter 6 performs a comparative evaluation of DiffVAE and Disentangled-DiffVAE with dimensionality reduction techniques commonly used on gene expression data.
- Chapter 7 discusses the work done in this project, suggests possible directions for future work and provides overall conclusions.

Chapter 2

Background

This chapter introduces neural networks, methods for training them, as well as the autoencoder architecture which forms the basis for the models designed in this project. To account for the multidisciplinary nature of the project, biological concepts about gene expression and haematopoietic stem cells will also be described, to provide an understanding of the data used and the results obtained.

2.1 Neural networks

Neural networks are defined as computational graphs consisting of interconnected artificial neurons that perform complex operations on the input data. The artificial neuron represents the fundamental unit of a neural network and it is illustrated in Figure 2.1.

The neuron computes a weighted sum of its input elements $\mathbf{x} = [x_1\, x_2 \ldots x_n]^T$ and applies an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ to the result:

$$y = \sigma\left(\sum_{i=1}^{n} x_i w_i + b\right) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right), \tag{2.1}$$

where $\mathbf{w}^T = [w_1\, w_2\, \ldots\, w_n]$ consists of the weights associated with each input element and b is the bias of the neuron.

The weight w_i indicates the importance of the i-th input x_i, while the bias

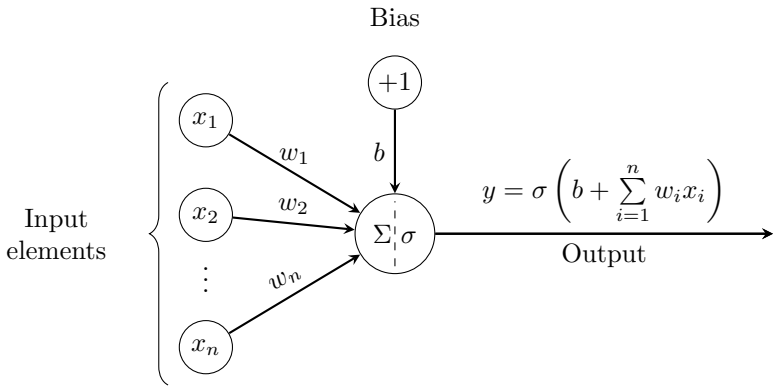


Figure 2.1: Structure of an artificial neuron: computational unit performing a weighted sum of its input elements and applying an activation function to the result to obtain the output.

b shifts the activation function on the horizontal axis. The activation function σ applied by an artificial neuron is generally non-linear, so that the neural network can learn to model complex functions. The choice of the activation function is highly dependent on the type of function the neural network needs to learn.

The computational graph of the neural network is obtained by arranging the artificial neurons in layers and by adding connections between them. Fully connected layers are obtained when each neuron in a given layer is connected to all of the neurons in the next layer, as illustrated in Figure 2.2.

Layer (l) in the neural network has n_l neurons and its activations are described by $\mathbf{x}^{(l)} = [x_1^{(l)}\, x_2^{(l)}\, \dots x_{n_l}^{(l)}]^T$. The weights from layer (l) and layer $(l+1)$ can be written compactly in a matrix form $\mathbf{W}^{(l)}$, where $\mathbf{W}_{ij}^{(l)}$ is the weight of the edge between neuron $x_i^{(l)}$ and $x_j^{(l+1)}$. Moreover, the biases of layer $(l+1)$ are given by $\mathbf{b}^{(l+1)} = [b_1^{(l+1)}\, b_2^{(l+1)}\, \dots\, b_{n_{l+1}}^{(l+1)}]^T$, where $b_i^{(l+1)}$ is the bias of neuron $x_i^{(l+1)}$.

The operations performed to compute the activation of neurons in layer $(l+1)$ are summarised by:

$$\mathbf{x}^{(l+1)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l+1)}\right). \tag{2.2}$$

A neural network can consist of several fully connected layers. For a network with L layers, the first layer $\mathbf{x}^{(1)}$ represents the input, the last layer $\mathbf{x}^{(L)}$ is the output and the layers $\mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \dots, \mathbf{x}^{(L-1)}$ are called hidden layers. A neural network with more than one hidden layer is referred to as a deep neural network.

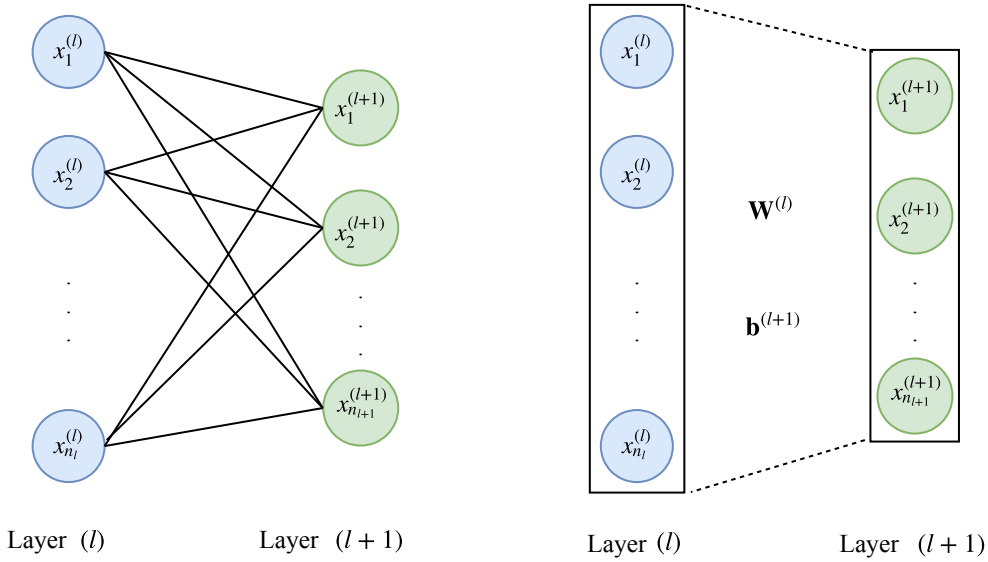


Figure 2.2: *Left:* Neurons in layer (l) are fully connected to neurons in layer $(l+1)$. The edges between neurons are parametrised by weights and each neuron has its own bias. *Right:* Compact representation of fully connected layers. Matrix $\mathbf{W}^{(l)}$ consists of the weights between neurons in layer (l) and neurons in layer $(l+1)$ and $b^{(l+1)}$ contains the biases of the neurons in layer $(l+1)$.

Cybenko [8] proved that a neural network with a single hidden layer consisting of a finite number of neurons can approximate any continuous function defined on a compact (closed and bound) subset of $\mathbb{R}^n$. While this **universal approximation theorem** does not provide a method for building such a network, it emphasises the computational capabilities of neural models.

2.2 Training neural networks

Let $\boldsymbol{\theta}$ be the set of all of the parameters in the neural network, i.e all of the weights $\mathbf{W}$ and biases $\mathbf{b}$. Then, $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ is the function described by the computational graph of the network under a particular setting for $\boldsymbol{\theta}$. For an input $\mathbf{x}$, let $\mathbf{y}' = f_{\boldsymbol{\theta}}(\mathbf{x})$ be the output of the network.

Given a training dataset $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots (\mathbf{x}_N, \mathbf{y}_N)\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in \mathbb{R}^m$, the parameters $\boldsymbol{\theta}$ in the networks can be adjusted, through training, such that the neural network learns to approximate a desired function $f_{\boldsymbol{\theta}^\star}$, where $\boldsymbol{\theta}^\star$ are the optimal set of parameters.

Neural networks are trained using gradient-based optimisation algorithms that aim to minimise a loss function. The loss function measures how well the network estimates each $\mathbf{y}_i$ from $\mathbf{x}_i$. Let $\mathcal{L}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$ be the loss under the parameter setting $\boldsymbol{\theta}$, input $\mathbf{x}$ and desired output $\mathbf{y}$. One example of such a loss function is the mean squared error, typically used for regression problems:

$$\mathcal{L}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) = \left(f_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}\right)^2. \tag{2.3}$$

Gradient descent minimises the loss function over the entire training set $\mathcal{T}$, by adjusting the parameters $\boldsymbol{\theta}$ in the direction of the steepest decrease in $\mathcal{L}$, as follows:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{T}} \frac{\partial \mathcal{L}(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i)}{\partial \boldsymbol{\theta}} \bigg|_{\boldsymbol{\theta}_t}, \tag{2.4}$$

where $\boldsymbol{\theta}_t$ are the parameters at iteration t and α is the learning rate, which decides how much of the gradient is subtracted from the parameters at each iteration.

The algorithm for training a neural network begins by initialising the weight and biases. Then, the following steps are iteratively performed until the loss converges:

- Use **forward propagation** to compute the output $\mathbf{y}'_i = f_{\boldsymbol{\theta}}(\mathbf{x})$ of the network for each $\mathbf{x}_i \in \mathcal{T}$; each $\mathbf{x}_i$ is given as input to the network and the layer activations are successively computed using equation 2.2 until the output layer is reached.

- Use **backpropagation** to compute $\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i)}{\partial \boldsymbol{\theta}}$ for each $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{T}$; at the output layer, the partial derivative $\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i)}{\partial \theta}$ can be directly computed from the loss function. Then, the results are backpropagated to the previous layer and the chain rule is used to compute the corresponding gradients. The process is repeated until the input layer is reached.

- Update the parameters in the network using equation 2.4.

In practice, **mini-batch gradient descent** is used to train neural networks. Thus, instead of performing the gradient updates over all of the N data points in the training dataset $\mathcal{T}$, they are only performed over a random sample of M data points drawn at each training iteration. Then, the neural network is trained for a number of **epochs** (full passes through all of the examples in the training dataset).

2.3 Autoencoders

Autoencoders are an application of neural networks to unsupervised learning and can learn useful representations from data, by using an architecture consisting of an encoder network and a decoder network, each defining a corresponding function:

Encoder: $enc : \mathbb{R}^n \to \mathbb{R}^m$
Maps the input $\mathbf{x} \in \mathbb{R}^n$ to a latent code $\mathbf{z} \in \mathbb{R}^m$: $\mathbf{z} = enc(\mathbf{x})$. The latent code is given by the activation of the neurons in the output layer of the encoder.

Decoder: $dec : \mathbb{R}^m \to \mathbb{R}^n$
Builds a reconstruction of the input $\mathbf{x}' \in \mathbb{R}^n$ from the code $\mathbf{z}$: $\mathbf{x}' = dec(z)$. The reconstruction is given by the output of the decoder.

An example of an autoencoder with a single hidden layer is given in Figure 2.3. The autoencoder's architecture is generally symmetric around the layer representing the code $\mathbf{z}$ and the function described by the model is $f_{\boldsymbol{\theta}}(\mathbf{x}) = dec(enc(\mathbf{x}))$.

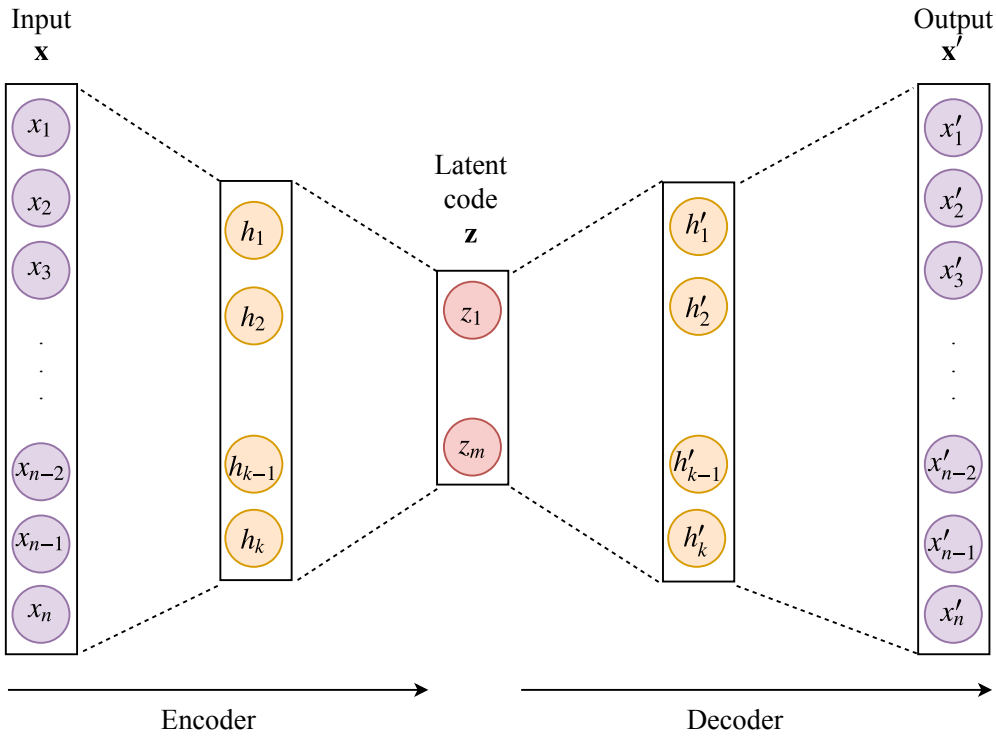


Figure 2.3: Simple autoencoder architecture: encoder learns to build a latent representation $\mathbf{z}$ for the input $\mathbf{x}$, while the decoder learns to build a reconstruction $\mathbf{x}'$ from this representation.

This simple autoencoder, represented more compactly in Figure 2.4 is usually trained to build a reconstruction $\mathbf{x}' = dec(enc(\mathbf{x}))$ that is as close as possible to the input $\mathbf{x}$, thus approximating the identity function. In order to do so, the model is trained to minimise the reconstruction error, which can be measured by the mean squared error loss function $\mathcal{L}(\mathbf{x}, \boldsymbol{\theta}) = \|\mathbf{x} - dec(enc(\mathbf{x}))\|^2$, where $\boldsymbol{\theta}$ consists of all of the parameters in the network.

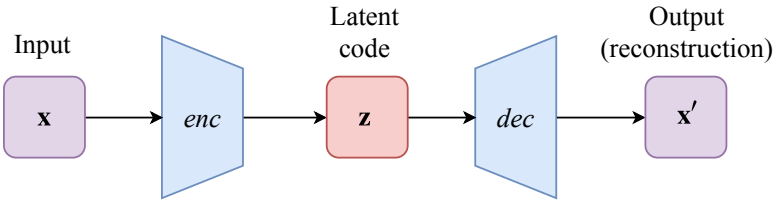


Figure 2.4: Compact representation of autoencoder mapping the input $\mathbf{x}$ to a reconstruction $\mathbf{x}'$ through a code $\mathbf{z}$.

The autoencoder can be used to perform non-linear dimensionality reduction, which is achieved when $m < n$, i.e the size of the latent code $\mathbf{z}$ is smaller than the size of the input $\mathbf{x}$. Such a model will be used in Chapter 6 as a baseline for comparison with the models developed in this project.

However, in order to build more meaningful representations from the input, the autoencoder architecture needs to be modified such that the model does not just approximate the identity function. Several autoencoder architectures have been developed to build better representations from data in an unsupervised manner. For instance, denoising autoencoders [9] can learn to reconstruct data corrupted by noise, and, at the same time, approximate the probability distribution of the observed data $p_{\text{data}}(\mathbf{x})$.

In this project, we will focus on the variational autoencoder, a model which learns to approximate the conditional probability distributions $p_{enc}(\mathbf{z}|\mathbf{x})$ and $p_{dec}(\mathbf{x}|\mathbf{z})$ through the encoder and decoder networks respectively. This model was chosen because cell differentiation can be best modelled stochastically [10, 11]. By modelling the generative process of the observed gene expression data, the variational autoencoder will prove useful in understanding the factors that lead to cell differentiation.

2.4 Gene expression and RNA sequencing

The process of gene expression involves converting the genetic information in the DNA into functional products such as proteins, functional RNA or other molecules which then dictate the cell function.

The development of high-throughput RNA sequencing (RNA-seq) technologies has made it possible to obtain gene expression measurement for whole transcriptomes. Recently, single-cell RNA sequencing (scRNA-seq), initially proposed by Tang *et al.*, became accessible due the decrease in sequencing costs and to the development of several protocols. An example of a common workflow for performing scRNA-seq is illustrated in Figure 2.5.

scRNA-seq allows the measurement of the distribution of expression levels for each gene across a population of cells, thus driving new research involving the understanding of the stochasticity of gene expression, heterogeneity of cell populations as well as the reconstruction of cellular development trajectories.

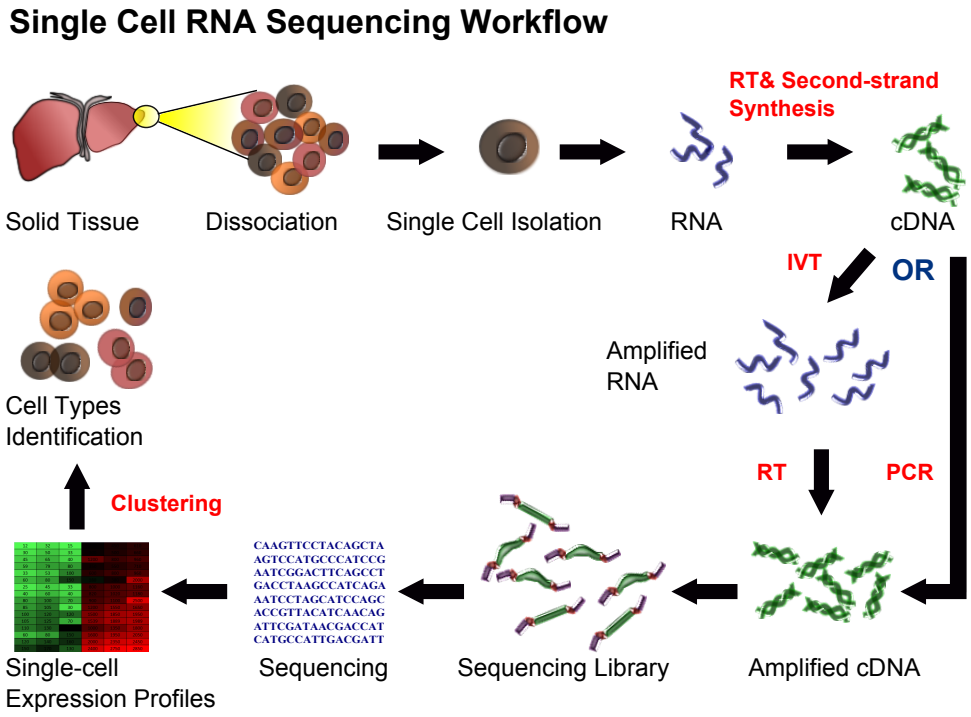


Figure 2.5: Example of single-cell RNA sequencing workflow: method used to obtain gene expression measurements for the cells. Image taken from Wikipedia [12].

The different cell types in a tissue are identified experimentally using cell surface markers or computationally using the Monocle [13, 14] algorithm, diffusion maps or clustering methods. Current computational methods applied to single-cell RNA-seq data rely on linear dimensionality reduction techniques [15, 16], which limits the potential of characterising individual cell types.

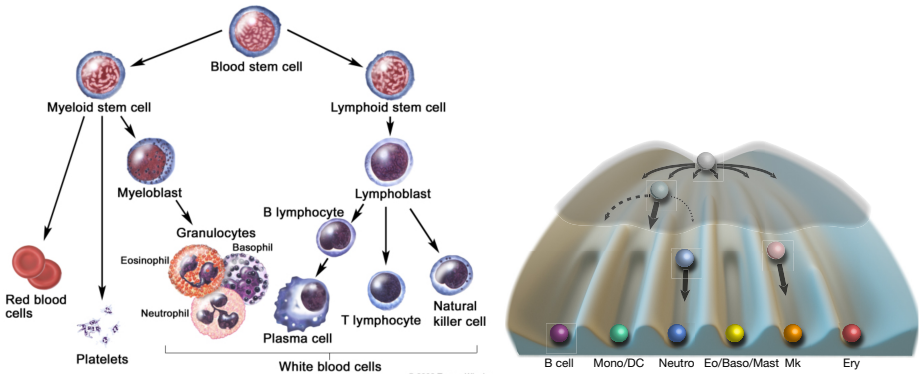
2.5 Hematopoietic stem cell differentiation

This project will develop methodology for applying machine learning methods to understand cell differentiation, with a case study done on haematopoietic (blood) stem cells. This section will describe the underlying motivation for choosing to analyse these type of cell.

Hematopoiesis is the process involving the formation, development and differentiation of blood cells. The hematopoietic stem cells are usually found in the bone marrow and they have the capacity to differentiate in any type of mature blood cell as well as self-renew and form more stem cells. The first step in the differentiation process gives rise to myeloid and lymphoid progenitor cells which can then follow differentiation trajectories resulting in specific blood cells. The progenitor cells cannot renew themselves.

Several possible differentiation trajectories (lineages) have been proposed for the formation of the different types of mature blood cells and two of such examples are illustrated in Figure 2.6. The hierarchical lineage has been used as a standard model, where at each step in the differentiation process, a cell can make several commitment choices, before becoming fully differentiated. Recent research has shown that lineage commitment is, in fact, a continuous process, where a stem or progenitor cell has a restricted possibility for changing its fate at the beginning of the differentiation process [17].

The cells at different steps in the differentiation process are characterised by different gene expression profiles. The blood types of cells are identified by clustering and generally, the intermediate steps through the differentiation process are also found through computational techniques.



(a) Hierarchical lineage (b) Continuous lineage

Figure 2.6: *Left:* Image taken from National Cancer Institute [18] illustrating a hierarchical lineage (differentiation trajectory) for a blood stem cell. At each step in the tree, the less specialised (progenitor) cell has several options for becoming more specialised. *Right:* Velten *et al.* [17] proposed a continuous lineage commitment for the blood stem cells where less specialised (progenitor) cells are more restricted in the type of mature blood cell they can become.

Machine learning methods have the potential to identify the transcriptomics factors (i.e. genes) that influence this differentiation process, as well as how stochastic changes in the gene expression profiles can give rise to different types of more specialised cells.

Chapter 3

Related work

This chapter briefly introduces the field of unsupervised deep learning and its applications in order to put the work into context. Then, we will discuss several unsupervised methods that have been applied to study gene expression data, as well as cell differentiation.

3.1 Unsupervised deep learning

Unsupervised learning methods are capable of extracting features from data without the need for a labelled training dataset. Deep learning methods distinguish themselves from other machine learning techniques through the fact that they decrease the need for manual feature engineering. Neural networks can successively apply non-linear operations to the input data in order to build abstract hierarchical features forming complex representation from the input data.

Several architectures for neural models have been proposed over time to perform unsupervised learning tasks. For instance, simple autoencoders have been used extensively for non-linear dimensionality reduction [4]. Conversely, Restricted Boltzmann Machines [19], Deep Belief Networks [20] and stacked autoencoders have been employed for unsupervised pre-training [21, 22] thus aiding classification tasks. Recently, variational autoencoders [5] and Generative Adversarial Networks [23] have been used as generative models for tasks such as dataset augmentation and data imputation.

These models have had many practical applications, improving machine learning tasks in fields such as:

Natural Language Processing Building distributed word representations (word embeddings) using the context of words in large corpora [24]) and obtaining shared representations across the tasks of language modelling, part-of-speech tagging and named entity recognition [25].

Computer Vision Learning visual representations from videos [26], learning useful features for image classification [21,27], predicting events in scenes [28] and image generation [29].

Bioinformatics Unsupervised pre-training for Micro-RNA target prediction [30] and mammographic risk scoring involving breast density segmentation [31], learning latent features and performing dimensionality reduction of transcriptomics data [32–34] and identifying accessibility of proteins [35].

Recently, there have been a large number of emerging deep learning architectures for analysing and finding patterns in graph-structured data [6, 36, 37]. Variational graph autoencoders, proposed by Kipf and Welling [7], have been effectively applied in unsupervised learning tasks such as link prediction in large scale relational data [38] and matrix completion for making recommendations [39].

This project leverages the success of the previously described unsupervised deep learning methods in order to develop appropriate architectures and novel methodologies that can be applied to analyse cell differentiation.

3.2 Methods for gene expression data

Gene expression data is complex and high dimensional: a single data point can consist of thousands of gene expression measurements. Building an appropriate low dimensional representation of the data is therefore crucial for being able to discover latent biological mechanisms, but also for being able to use this data in subsequent analysis steps, such as classification or clustering.

Previous approaches for finding good representations from high-dimensional biological data are based on linear dimensionality reduction methods such as principal component analysis (PCA) [40]. However, such linear methods, are not able to capture the complex relationships between the input dimensions and disregard

meaningful information in the data. In addition, Yeung and Ruzzo [41] showed that using PCA before clustering gene expression data has a negative effect on the quality of the clusters. Despite these findings, a lot of research in gene expression analysis [42–44] is based on applying PCA before clustering cells to identify their types (last step in Figure 2.5).

While kernel techniques can make PCA non-linear [45–47], they are highly dependent on finding a suitable similarity matrix between data points or a feature space where naive similarity metrics can be used. A much better solution involves learning features from the data, a task at which deep learning models excel.

Autoencoders can be used to perform non-linear dimensionality reduction, but also to extract biologically relevant latent features from transcriptomics (gene expression) data. Related work in this area shows the effectiveness of these models in analysing gene expression data:

Tybalt Way *et al.* [32] trained a variational autoencoder on pan-cancer RNA-seq data from The Cancer Genome Atlas (TCGA) [48] to explore the biological relevance of the latent space produced by the autoencoder. The results indicated that the latent dimensions in the variational autoencoder could be used to distinguish the patient gender, as well as patterns of metastatic activation.

ADAGE Tan *et al.* [33] built a denoising autoencoder capable of modelling the response of cells to low oxygen and finding differences between strains in gene expression from Pseudomonas aeruginosa.

Dhaka. Rashid *et al.* [34] developed a variational autoencoder model to identify tumour subpopulations, marker genes, as well as differentiation trajectories for the malignant cells using scRNA-seq genomic data. Moreover, a comparison is made between Dhaka and other dimensionality reduction techniques including PCA and t-distributed Stochastic Neighbor Embedding to show the ability of the autoencoder to extract better latent features. However, Dhaka has a latent representation of only 3-dimensions which does not encode by itself biologically relevant information.

This project improves on the existing methods based on autoencoders for studying gene expression data by building on methodology for analysing the latent representation in the context of modelling cell differentiation. Moreover, previous work does not utilise ideas from information theory, which can be used to disentangle the latent representation of the autoencoder in order to reveal more sources of variation in the observed. Another novelty if given by using a variational graph-autoencoder autoencoder to predict links between cells.

Chapter 4

Design and implementation

This chapter discusses the design and implementation of DiffVAE, Disentangled-DiffVAE and Graph-DiffVAE, the unsupervised neural architectures proposed to model cell differentiation.

The chapter begins by describing the probabilistic framework built by the variational autoencoder and how it can be used to reason about cell differentiation. Then, the specific architecture chosen for DiffVAE will be presented, as well as the disentanglement methods applied to achieve better separation of the factors of variation in the data through Disentangled-DiffVAE. Finally, we shall explore how spectral graph convolutions can incorporated in the variational autoencoder to predict links between cells.

4.1 Variational autoencoder

The variational autoencoder, proposed by Kingma and Welling [5], captures the essential characteristics of the input observations by modelling the latent code $\mathbf{z}$ as a continuous random variable thus building a generative model. Through training, the variational autoencoder learns the probability distribution of $\mathbf{z}$.

The variational autoencoder was chosen over simpler autoencoder models because the changes in gene expression causing less specialised cells to differentiate into more specialised ones are best modelled stochastically [10,11]. More specifically, the latent representation of the variational autoencoder can probabilistically model the biological processes that caused the changes in the observed gene expression for the

cells in the dataset.

This is achieved by building a probabilistic framework where a biomedical dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ with gene expression measurements for the different cells consists of N i.i.d samples, also known as observations, from a continuous random variable $\mathbf{x}$, having the distribution $p_{\text{data}}(\mathbf{x})$. The gene expression data is assumed to be generated by some random process, modelled by an unobserved continuous random variable $\mathbf{z}$.

In this Bayesian framework, the latent codes $\mathbf{z}$ have a parametrised prior distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$ and the random process for generating the data is given by sampling from the prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ and then from the parametrised conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. In our case, the parameters $\boldsymbol{\theta}$ will be represented by the weights in a neural network.

The values of $\mathbf{z}$ for the training dataset and the true parameters $\boldsymbol{\theta}$ are unknown. One option for learning in this generative framework is to maximise the marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$ using the observations from the training dataset. The marginal likelihood, also known as the evidence, is computed by integrating over the possible codes:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int_{\mathbf{z}\in\mathcal{Z}} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\mathrm{d}\mathbf{z} = \int_{\mathbf{z}\in\mathcal{Z}} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})\mathrm{d}\mathbf{z}. \tag{4.1}$$

Computing the integral involves spanning the whole space of values for $\mathbf{z}$ which is often intractable, particularly in the case of neural networks with non-linear activations. Additionally, applying Monte Carlo methods to approximate the integral above is practically infeasible when large datasets are involved because $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is slow to compute.

To perform inference, the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ has to be computed. The posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is needed for building the latent representation from the input data. Nevertheless, computing this posterior is also intractable. Attempting to use Bayes theorem gives us:

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})}, \tag{4.2}$$

where the marginal likelihood $p_{\theta}(\mathbf{x})$ appears again in the denominator.

Variational inference also known as Variational Bayes is a method for approximating the posterior distribution in situations such as the ones described in this project, where it cannot be computed analytically. In variational inference, a

recognition model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is introduced and is used to approximate the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The recognition model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is parametrised by $\boldsymbol{\phi}$, which in this case will also be represented by parameters in a neural network.

The variational autoencoder, builds an **inference (recognition) model** $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ in the encoder network and a **generative model** $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ in the decoder network, as illustrated in Figure 4.1.

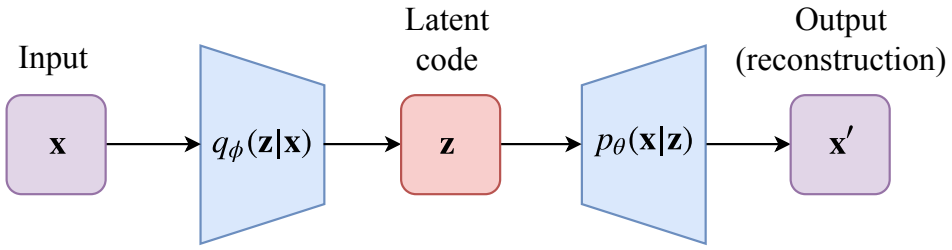


Figure 4.1: Compact representation of the variational autoencoder. The encoder builds a recognition model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ used to approximate the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, while the decoder builds a generative model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ for the data.

The parameters in the variational autoencoder are trained using gradient-based optimisation. For this, an objective function needs to be decided. Given the design of the variational autoencoder, its objective can be split into two parts:

- the encoder network needs to minimise the divergence between the recognition model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The Kullback–Leibler divergence D_{KL} can used, which is defined as follows:

$$D_{KL}(Q\|P) = \int_{-\infty}^{+\infty} q(x)\log\frac{q(x)}{p(x)}\mathrm{d}x, \tag{4.3}$$

  where Q and P are two continuous probability distributions, with densities $q(x)$ and $p(x)$ respectively. $D_{KL}(Q\|P) \geq 0$ with equality obtained only if $Q = P$.

  Thus, a good objective for the encoder would be:

$$\text{minimise } D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}));$$

- the decoder network needs to be able to generate with high probability the training examples. This can be achieved by maximising the marginal log-

likelihood of observing the data points in the training dataset. Therefore, the decoder's objective is:

$$\text{maximise } \log p_{\boldsymbol{\theta}}(\mathbf{x}).$$

By combining these two objectives, we obtain the evidence lower bound (ELBO), which the network is trained to maximise:

$$\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})). \tag{4.4}$$

The $\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi})$ represents a lower bound of the model evidence given by the log-likelihood: $\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) \leq \log p_{\boldsymbol{\theta}}(\mathbf{x})$, with equality achieved when the recognition model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ described by the encoder approximates the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ perfectly.

In its current form, the $\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi})$ cannot be evaluated, as it still needs $p_{\boldsymbol{\theta}}(\mathbf{x})$ for the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ to be computed. However, it can be rewritten as follows (see Appendix A.1 for a full derivation):

$$\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left(\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right)}_{\text{Reconstruction accuracy}} - \underbrace{D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}))}_{\text{Regularisation}}. \tag{4.5}$$

The variational autoencoder defines an objective function $\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$ in terms of the $\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi})$. Gradient-based optimisation is then applied to adjust the parameters in the encoder $\boldsymbol{\phi}$ and decoder $\boldsymbol{\theta}$ to maximise the objective function using the observations $\mathbf{x}$ in our dataset.

4.1.1 Variational autoencoder architecture

The variational autoencoder assigns a isotropic multivariate Gaussian prior to the latent variables: $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. Moreover, the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is also assumed to have an approximate Gaussian form. Thus, the recognition model $\mathbf{q}_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is approximated with a multivariate Gaussian $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$.

The encoder network is constructed to model the diagonal multivariate Gaussian $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ used to approximate $\mathbf{q}_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. Hence, two fully connected layers form the output of the encoder, one for computing the mean $\boldsymbol{\mu} = [\mu_1\, \mu_2\, \ldots\, \mu_m]$ and another one for computing the variance $\boldsymbol{\sigma}^2 = [\sigma_1^2\, \sigma_2^2\, \ldots\, \sigma_m^2]$, where m is the number of latent dimensions used in the model.

The latent representation is given by $\mathbf{z} = [z_1\, z_2\, \ldots\, z_m]^T$, where each latent variable z_i is sampled from its conditional distribution: $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. Then, the decoder network consists of several layers that are used to reconstruct $\mathbf{x}$ from $\mathbf{z}$, by modelling $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

An example of a variational autoencoder consisting of one hidden layer in both the encoder and decoder is illustrated in Figure 4.2.

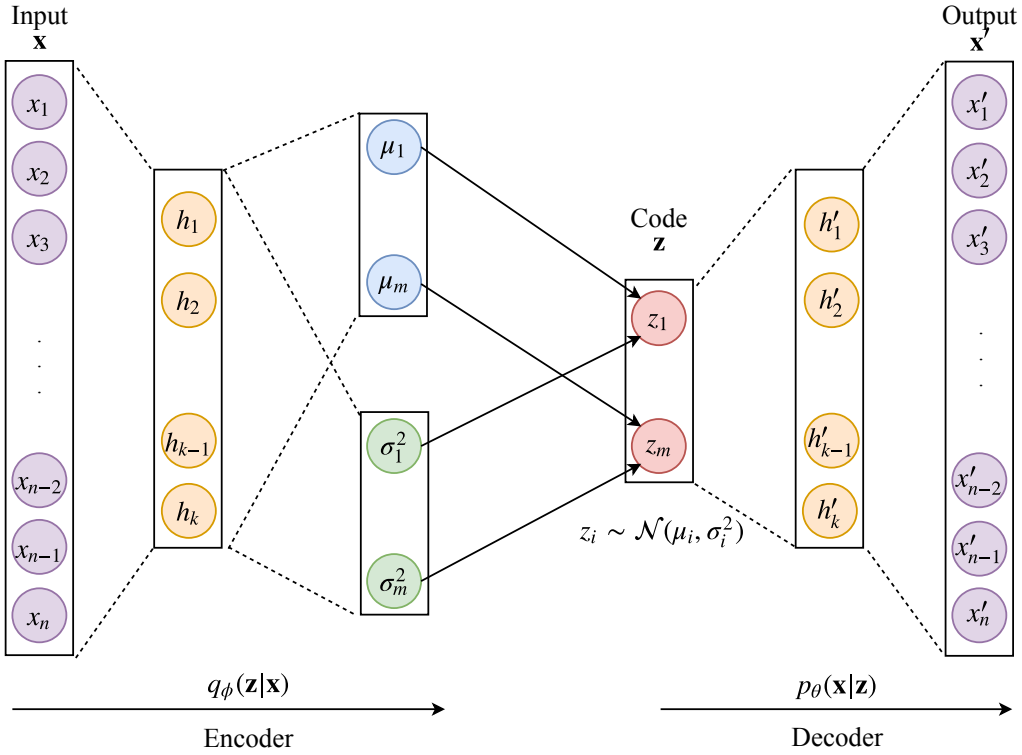


Figure 4.2: Architecture of a variational autoencoder consisting of fully-connected layers in the encoder and decoder. The encoder computes the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$ of the Gaussian distribution that can be used to sample the latent code $\mathbf{z}$. The sampling of the elements of $\mathbf{z}$ is indicated by the continuous arrows. The decoder generates the output from the latent code by approximating $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

4.1.2 Reparameterisation trick

The stochastic operations added in the network to sample the latent code $\mathbf{z}$ cause problems to the standard gradient-based algorithm, as it is not possible to compute gradients through the random sampling of $\mathbf{z}$.

To overcome these issues, Kingma and Welling [5] proposed the reparameterisation trick that involves parameterising the latent code as follows:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\varepsilon} \odot \boldsymbol{\sigma}, \tag{4.6}$$

where $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is an auxiliary variable, and $\odot$ is the element-wise (Hadamard) product. This method maintains the probability distribution of $\mathbf{z}$ and it enables the computation of gradient with respect to the loss function for all the parameters in the network. Figure 4.3 shows the changes made to perform the reprameterisation trick.

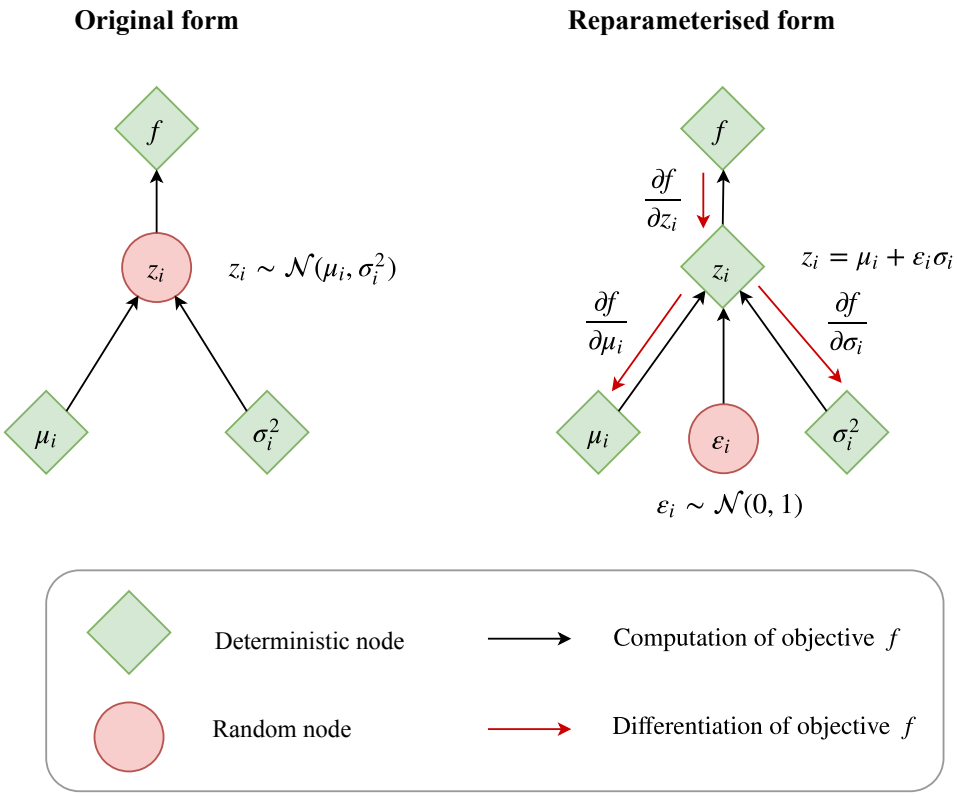


Figure 4.3: Reparameterisation trick used in the variational autoencoder. In the original form, each z_i is directly sampled from $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. This prevents the gradients of the objective f to be computed with respect to the parameters in the encoder since it is not possible to do backpropagation through the random node z_i. In the reparameterised form, the auxiliary random variable $\varepsilon \sim \mathcal{N}(0, 1)$ is introduced, thus enabling the parameterisation $z_i = \mu_i + \varepsilon_i \sigma_i$. This makes z_i a deterministic node and allows backpropagation through z_i to compute the gradient of f with respect to the parameters in the encoder.

4.1.3 Objective function for the variational autoencoder

The objective function maximised by the variational autoencoder is based on the ELBO($\boldsymbol{\theta}, \boldsymbol{\phi}$) in equation 4.5 and is given by:

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}(\mathbf{z}|\mathbf{x})}}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})). \tag{4.7}$$

Regularisation term: $-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}))$.

Using the assumptions made on the prior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(z; \mathbf{0}, \mathbf{I})$ and on the recognition model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ (described in section 4.1.1), the KL divergence term can be computed analytically as follows:

$$- D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2\mathbf{I})) \| \mathcal{N}(\mathbf{0}, \mathbf{I})) = \frac{1}{2} \sum_{i=1}^{m} (1 + \log(\sigma_i^2) - \mu_j^2 - \sigma_j^2), \tag{4.8}$$

where m is the number of latent dimensions in $\mathbf{z}$.

Reconstruction accuracy: $\mathbb{E}_{q_{\boldsymbol{\phi}(\mathbf{z}|\mathbf{x})}}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$.

Attempting to compute the reconstruction accuracy by integrating over all possible values of $\mathbf{z}$,

$$\mathbb{E}_{q_{\boldsymbol{\phi}(\mathbf{z}|\mathbf{x})}}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] = \int_{\mathbf{z} \in \mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})) \, \mathrm{d}\mathbf{z}, \tag{4.9}$$

is typically intractable. The reconstruction accuracy can be estimated using Monte-Carlo sampling:

$$\mathbb{E}_{q_{\boldsymbol{\phi}(\mathbf{z}|\mathbf{x})}}(\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})) = \frac{1}{L} \sum_{i=1}^{L} \log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_i)), \tag{4.10}$$

where $\mathbf{z}_i \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and L is the number of samples taken per data point.

When using mini-batches for training, a sample of B data points from the full dataset is used to perform the gradient updates (see Section 2.2). In this situation, Kingma and Welling [5] found experimentally that when B is large enough (e.g. $B = 100$) having one sample ($L = 1$) is sufficient to estimate the reconstruction accuracy.

Thus, the average minibatch objective is given by:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^B) = \frac{1}{B} \sum_{\mathbf{x} \in \mathbf{X}^B} \left(\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \frac{1}{2} \sum_{j=1}^{m} (1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)) \right), \quad (4.11)$$

where the $\mathbf{X}^B = \{\mathbf{x}^{(i)}\}_{i=1}^B$ is a mini-batch with B samples randomly drawn from the full dataset $\mathcal{D}$.

4.2 DiffVAE

The variational autoencoder (VAE) architecture proposed in this project was built to model the differentiation of cells, and it was consequently named DiffVAE. This section will describe the implementation details of DiffVAE, as well as the choices made in designing the model.

4.2.1 Architecture details

DiffVAE was constructed such that both the encoder and decoder consists of two fully connected hidden layers. The incorporation of multiple hidden layers helps to build a hierarchical representation of features, thus obtaining a more complex model. The size of the hidden layers is symmetric between the encoder and decoder.

An important design choice is made when selecting the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ applied by the fully connected layers. Figure 4.4 illustrates common activation functions: the logistic function ($\sigma(x) = \frac{1}{1+\exp(-x)}$), hyperbolic tangent ($\sigma(x) = \tanh(x)$), rectified linear unit [49] ($\sigma(x) = \text{ReLU}(x) = \max(0, x)$) and the linear function $\sigma(x) = x$. Using a non-linear activation function is essential for being able to build a good representation from the complex gene expression data describing the cells. Moreover, the gradient of the activation function is also crucial in ensuring convergence of the training algorithm.

As is commonly employed in neural networks, the ReLU activation will be applied in the hidden layers of both the encoder and decoder in order to introduce non-linearity in the network. The non-saturating gradient of this activation function, compared to the gradients of the logistic or tanh functions will also help the training algorithm to converge more quickly [1].

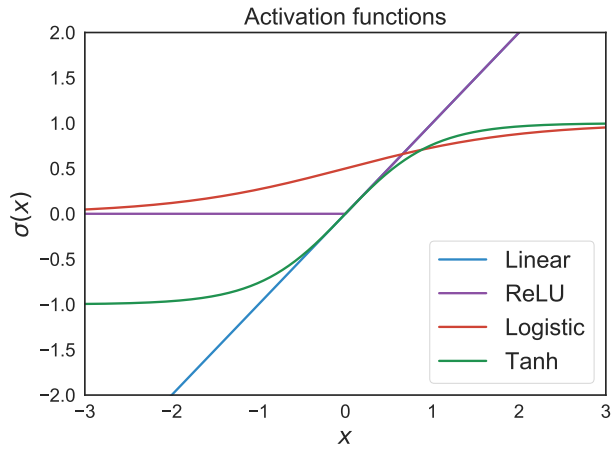


Figure 4.4: Common activation functions used in neural network architectures.

The encoder and decoder networks of DiffVAE can be summarised in terms of the inference and recognition models they build.

Inference model (encoder): $\mathbf{q}_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$

The encoder consists of fully connected layers and has a Gaussian output. For numerical stability, the encoder network learns $\log(\boldsymbol{\sigma}^2)$ instead of $\boldsymbol{\sigma}^2$. The input to the encoder is $\mathbf{x}$, which, in our case, represents the gene expression data. The operations performed by the encoder network are summarised by:

$$\mathbf{x}_{\text{enc}}^{(1)} = \text{ReLU}(\mathbf{W}_{\text{enc}}^{(0)}\mathbf{x} + \mathbf{b}_{\text{enc}}^{(1)}), \tag{4.12}$$

$$\mathbf{x}_{\text{enc}}^{(2)} = \text{ReLU}(\mathbf{W}_{\text{enc}}^{(1)}\mathbf{x}_{\text{enc}}^{(1)} + \mathbf{b}_{\text{enc}}^{(2)}), \tag{4.13}$$

$$\boldsymbol{\mu} = \text{ReLU}(\mathbf{W}_{\mu}\mathbf{x}_{\text{enc}}^{(2)} + \mathbf{b}_{\mu}), \tag{4.14}$$

$$\log \boldsymbol{\sigma}^2 = \text{ReLU}(\mathbf{W}_{\sigma}\mathbf{x}_{\text{enc}}^{(2)} + \mathbf{b}_{\sigma}). \tag{4.15}$$

Sampling of $\mathbf{z}$:

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{4.16}$$

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\varepsilon} \odot \boldsymbol{\sigma}. \tag{4.17}$$

The trainable parameters in the encoder are:

$$\boldsymbol{\phi} = \{\mathbf{W}_{\text{enc}}^{(0)}, \mathbf{b}_{\text{enc}}^{(1)}, \mathbf{W}_{\text{enc}}^{(1)}, \mathbf{b}_{\text{enc}}^{(2)}, \mathbf{W}_{\mu}, \mathbf{b}_{\mu}, \mathbf{W}_{\sigma}, \mathbf{b}_{\sigma}\}.$$

Generative model (decoder): $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$

The output of the decoder has to reward the likelihood of the data we want to generate with this model. In our case, for each data point, the gene expression values can be modelled as samples from a multivariate Bernoulli distribution. Intuitively, each input gene is modelled as a Bernoulli random variable, and a sample from this distribution indicates whether the gene is expressed or not.

To build a decoder with Bernoulli output, we need to apply the logistic activation function to compute the output of the decoder because it takes values in the range $[0, 1]$.

The input to the decoder is the latent representation $\mathbf{z}$. The decoder performs the following operations in order to obtain the reconstructed input $\mathbf{x}'$:

$$\mathbf{x}_{\text{dec}}^{(1)} = \text{ReLU}(\mathbf{W}_{\text{dec}}^{(0)}\mathbf{z} + \mathbf{b}_{\text{dec}}^{(1)}), \tag{4.18}$$

$$\mathbf{x}_{\text{dec}}^{(2)} = \text{ReLU}(\mathbf{W}_{\text{dec}}^{(1)}\mathbf{x}_{\text{dec}}^{(1)} + \mathbf{b}_{\text{dec}}^{(2)}), \tag{4.19}$$

$$\mathbf{x}' = \sigma(\mathbf{W}_{\text{out}}\mathbf{x}^{(2)} + \mathbf{b}_{\text{out}}), \tag{4.20}$$

where σ is the logistic activation function. As $\mathbf{x}'$ is not sampled, we provide a maximum likelihood estimate for the reconstruction.

The trainable parameters in the decoder are:

$$\boldsymbol{\theta} = \{\mathbf{W}_{\text{dec}}^{(0)}, \mathbf{b}_{\text{dec}}^{(1)}, \mathbf{W}_{\text{dec}}^{(1)}, \mathbf{b}_{\text{dec}}^{(2)}, \mathbf{W}_{\text{out}}, \mathbf{b}_{\text{out}}\}.$$

Reconstruction accuracy for DiffVAE

To compute the reconstruction accuracy for DiffVAE, we make the assumption that the output (gene expression values) are independent. Then, using the input $\mathbf{x} = [x_1\, x_2\, \ldots\, x_n]$ and output $\mathbf{x}' = [x_1'\, x_2'\, \ldots\, x_n']$, where each $x_i' = p_{\boldsymbol{\theta}}(x_i = 1|\mathbf{z})$ we can write the multivariate Bernoulli distribution $\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ as follows:

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(x_i|\mathbf{z}) = \prod_{i=1}^{n} x_i'^{x_i}(1 - x_i')^{1-x_i}. \tag{4.21}$$

Then, the reconstruction accuracy of the DiffVAE, given by $\log\left(\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right)$ can

be computed using equation 4.21, as follows:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{n} \log p_{\theta}(x_i|\mathbf{z}) \tag{4.22}$$

$$= \sum_{i=1}^{n} x_i \log x_i' + (1 - x_i) \log (1 - x_i'). \tag{4.23}$$

The sizes of hidden layers and of the latent space were determined using cross-validation, as described in Section 4.2.4. The final architecture of DiffVAE is illustrated in Figure 4.5.

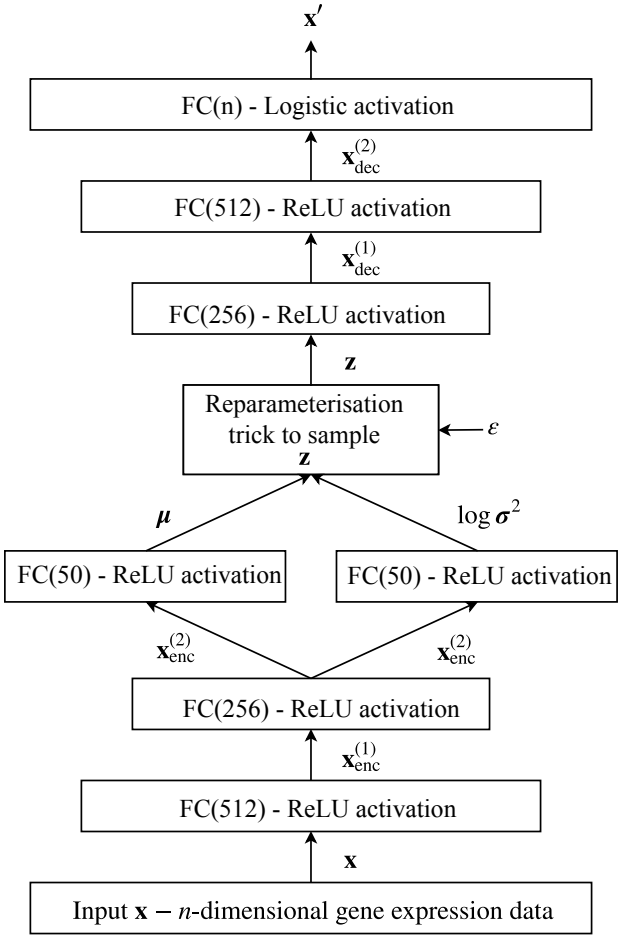


Figure 4.5: Final neural architecture for DiffVAE. Both the encoder and decoder consist of several fully-connected (FC) layers of sizes specified in brackets which use the corresponding activation function.

4.2.2 Training details

The training algorithm adjusts the parameters in DiffVAE such that the encoder learns to approximate the intractable posterior while the decoder learns to maximise the likelihood. To do so, minibatch gradient-based optimisation is used, where the final form of the mini-batch average objective function that is maximised is:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^B) = \frac{1}{B} \sum_{\mathbf{x} \in \mathbf{X}^B} \left(\sum_{i=j}^{n} x_i \log x_i' + (1 - x_i) \log (1 - x_i') + \frac{1}{2} \sum_{i=j}^{m} (1 + \log(\sigma_i)^2 - \mu_j^2 - \sigma_j^2) \right). \tag{4.24}$$

In order to perform the gradient descent optimisation described in Section 2.2, the negative value of the objective function $-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^B)$ will be instead minimised. In this context, $-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^B)$ will represent the loss function.

Minimising $-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^B)$ represents a non-convex optimisation problem where the aim of the gradient descent algorithm is to converge to a local minima. The initialisation of the weights and biases is important for converge, as well as for the quality of the local minima obtained. An appropriate initialisation methods assign random values to the weights in order to break the initial symmetry in the network.

Ideally, the initial distribution of the weights should help the signal reach the deeper layers of the network. One option for doing so is by using Xavier initialisation [50], which assign initial values to the weights from layer (l) to layer $(l+1)$ from a Uniform distribution as follows:

$$\mathbf{W}^{(l)} \sim \mathcal{U}[-\frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}}, \frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}}], \tag{4.25}$$

where n_l is the number of neurons in layer (l) and $n_{(l+1)}$ is the number of neurons in layer $(l+1)$. Xavier initialisation ensures that the variance of the weights is $\text{Var}(\mathbf{W}^{(l)}) = \frac{2}{n_l + n_{l+1}}$, which helps with vanishing/exploding gradients [50].

The learning rate decides how much of the gradient should be subtracted from the parameters at each training iteration and thus also has an important role in convergence. Figure 4.6 illustrates this idea by showing how different settings for the learning rate can affect convergence. The initial value of the learning rate is an hyperparameter, whose value is set as specified in Section 4.2.4.

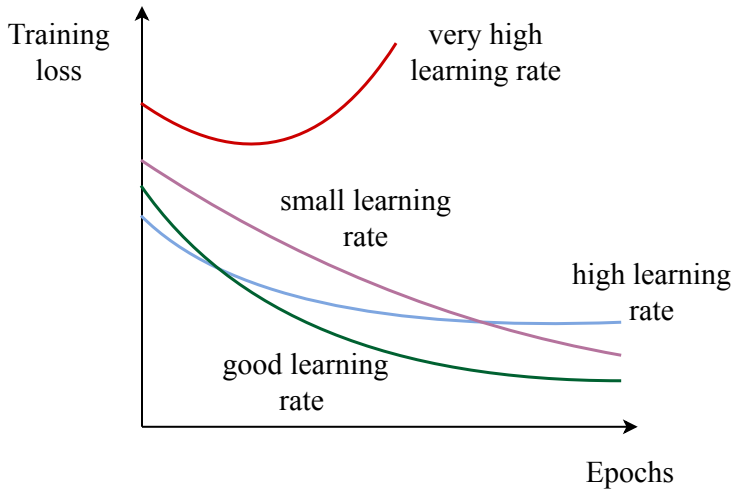


Figure 4.6: Illustration of how different settings of the learning rate affect the convergence of the gradient descent algorithm. Figure adapted from [51].

DiffVAE is trained using the **Adam optimiser** [52] which computes adaptive learning rates for each parameter in the network. The Adam optimiser achieves this by using of the momentum method [53] to accumulate the sum of gradients over a window through an exponentially decaying average of all past gradients (first moment) and square gradients (second moment).

4.2.3 Batch normalisation

One problem that can arise during training is the internal covariate shift, which happens when the distributions of inputs to the hidden layers have significant discrepancies. A solution to this problem is given by batch normalisation, a method proposed by Ioffe and Szegedy [54]. Batch normalisation involves normalising the input values to every layer of the neural network across a mini-batch.

Let $\{\mathbf{x}_1, \dots \mathbf{x}_B\}$ be the input to a layer in the network over a mini-batch $\mathcal{B}$. Let $x_{ij} = (\mathbf{x}_i)_j$ be the j-th value of $\mathbf{x}_i$. The mean μ_j and variance σ_j^2 of the j-th input over mini-batch $\mathcal{B}$ are given by:

$$\mu_j = \frac{1}{B}\sum_{i=1}^{B} x_{ij}, \qquad \sigma_j^2 = \frac{1}{B}\sum_{i=1}^{B} \left(x_{ij} - \mu_j\right)^2. \tag{4.26}$$

The normalised inputs are computed independently for each i, j as follows:

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}, \tag{4.27}$$

However, only using this re-standardisation restricts the input range to a layer in the network, and affects the model's representation capabilities. To overcome this issue, batch normalisation allows the model to fall back to the original values by learning scale γ and shift β parameters for $\hat{x}_{ij}$. Thus, the final normalised value y_{ij} is computed as:

$$y_{ij} = \gamma \hat{x}_{ij} + \beta. \tag{4.28}$$

Figure 4.7 illustrates the transformation applied to the inputs to layer $(l+1)$ during training in order to normalise their values across the batch $\mathcal{B}$. During testing, the normalised inputs are computed using the population statistics and the trained scale and shift parameters.

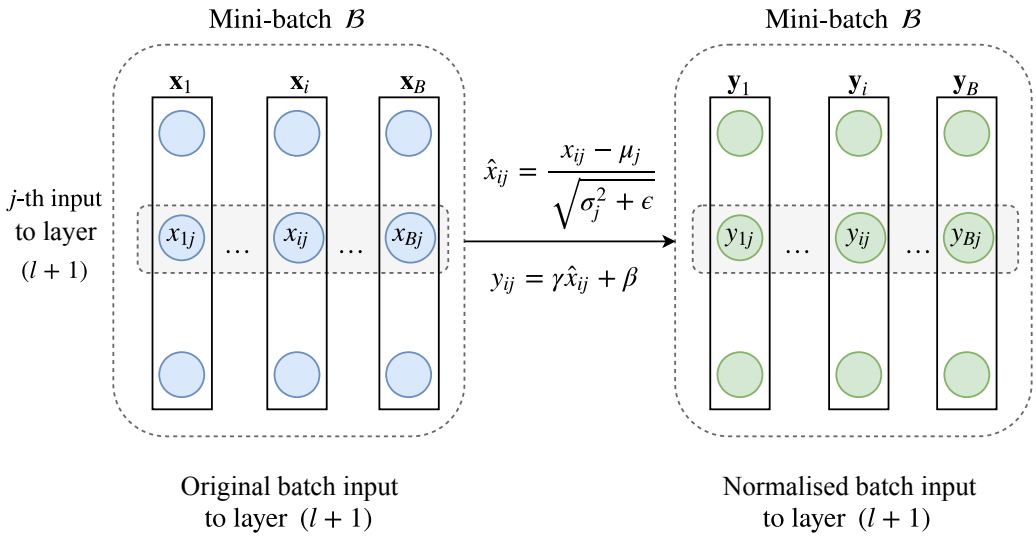


Figure 4.7: Batch normalisation transformation applied to the inputs to layer $(l+1)$ in a neural network. Each input y_{ij} is normalised using the mini-batch statistics (μ_j and σ_k) and then shifted and scaled through the learnt parameters γ and β.

4.2.4 Hyperparameter setting

The models developed include a high number of hyperparameters such as (but not limited to) the initial setting for the learning rate, the batch size, the number of neurons in the hidden layers. These hyperparameters were optimised by using cross-validation.

The original dataset was split such that 80% of the data points were used for training and 20% for validation. Each hyperparameter was optimised for a range of possible values, with the final one chosen based on the validation loss. The final values set in each architecture for all of the hyperparameters are summarised in Table 4.1:

Hyper-parameter	Value setting
Encoder layers	FC(512), FC(256)
Decoder layers	FC(256), FC(512)
Latent size	50
Learning rate	0.001
Batch size	128
Training epochs	100
Optimiser used	Adam

Table 4.1: Hyperparameter setting for DiffVAE. Fully connected (FC) layers are described by their size, e.g. FC(512) indicates a layer consisting of 512 neurons.

4.3 Disentangled-DiffVAE

In this section, we shall explore how the objective of the variational autoencoder can be improved in order to maximise the mutual information between the input data and the latent dimension. The model trained using this changed objective is named Disentangled-DiffVAE, as it is able to build a data representation where the latent biological factors influencing cell differentiation are better separated.

4.3.1 Variational autoencoder objective revisited

The ELBO objective that the variational autoencoder is trained to maximise has the following form:

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z})). \tag{4.29}$$

By analysing the divergence term $-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}))$ we notice that the ELBO objective is encouraging the posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ computed for each input $\mathbf{x}$ to have a distribution similar to the prior $p_{\boldsymbol{\theta}}(\mathbf{z})$.

When training DiffVAE, the objective is computed in expectation over the training data, i.e. the gene expression levels for the different cells in the datasets, which are samples from the distribution $p_{\text{data}}(\mathbf{x})$. We can show that (see A.2 for full derivation) [55, 56]:

$$\begin{aligned} \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}))] &= I(\mathbf{x};\mathbf{z}) + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z})||p_{\boldsymbol{\theta}}(\mathbf{z})) && (4.30)\\ &\geq I(\mathbf{x};\mathbf{z}), && (4.31) \end{aligned}$$

where $I(\mathbf{x};\mathbf{z})$ is the mutual information between the input $\mathbf{x}$ and latent representation $\mathbf{z}$ and $q_{\boldsymbol{\phi}}(\mathbf{z}) = \mathbb{E}_{p_{\text{data}}(\mathbf{z})}[q_{\phi}(\mathbf{z}|\mathbf{x})]$.

Hence, using a training objective to maximise $-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}))$ penalises the mutual information between the input and the latent representation [57]. Consequently, the latent dimension of the variational autoencoder can become uninformative for the input distribution.

Higgins *et al.* [58] propose adding a positive β coefficient to the KL divergence term $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$. Even though experiments have shown that having a $\beta \geq 1$ improves disentanglement and interpretability of latent dimensions [58, 59], it does not solve the problem mentioned above. On the contrary, it will make the latent

code even more uninformative, as a large value for β will encourage the latent code $\mathbf{z}$ to be independent of $\mathbf{x}$.

A solution to this problem was proposed by Zhao *et al.* [60] who introduce the InfoVAE family of autoencoders that have the following objective:

$$\mathcal{L}_{\text{InfoVAE}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - \lambda D(q_{\boldsymbol{\phi}}(\mathbf{z}) \| p_{\boldsymbol{\theta}}(\mathbf{z})), \tag{4.32}$$

where D can be any strict divergence function satisfying $D(q\|p) > 0$ and $D(q||p) = 0$ if the probability distributions p and q are the same, and where $\lambda > 0$ is a scaling coefficient.

Zhao *et al.* [60] prove that InfoVAE maximises the mutual information between the input and the latent representation. Moreover, minimising the divergence between $D(q_{\boldsymbol{\phi}}(\mathbf{z}) \| p_{\boldsymbol{\theta}}(\mathbf{z}))$, will encourage $q_{\boldsymbol{\phi}}$ to be similar to the prior $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ with diagonal covariance matrix, which will lead to disentanglement in the latent dimension.

4.3.2 Disentangled-DiffVAE details

Disentangled-DiffVAE is part of the InfoVAE family of variational autoencoders. As previously described, this family of autoencoders has the capacity to learn disentangled representations in the latent space, hence the name Disentangled-DiffVAE for the model architecture proposed to understand cell differentiation. Disentanglement is desirable in our case because, ideally, the latent representation built by the autoencoder should be able to separate the biological factors that have led to the development of various cell types.

The divergence measure used by Disentangled-DiffVAE is the maximum mean discrepancy (MMD) [61–63] between $q_{\boldsymbol{\phi}}(\mathbf{z})$ and $p_{\boldsymbol{\theta}}(\mathbf{z})$. Thus, Disentangled-DiffVAE is a type of MMD-VAE, a variational autoencoder model also proposed by Zhao *et al.* [60].

The MMD divergence measures how different the moments of two probability distributions are. The intuition behind the MMD divergence is given by the fact that two probability distributions are identical if and only if their moments match.

The MMD divergence compares the moments of $q_{\boldsymbol{\phi}}(\mathbf{z})$ and $p_{\boldsymbol{\theta}}(\mathbf{z})$, and it can be implemented through the use of the kernel trick:

$$\begin{aligned}\text{MMD}(q_{\boldsymbol{\phi}}(\mathbf{z})\|p_{\boldsymbol{\theta}}(\mathbf{z})) = \mathbb{E}_{p(\mathbf{z}),p(\mathbf{z}')}[k(\mathbf{z},\mathbf{z}')] + \mathbb{E}_{q(\mathbf{z}),q(\mathbf{z}')}[k(\mathbf{z},\mathbf{z}')] \\ - 2\mathbb{E}_{p(\mathbf{z}),q(\mathbf{z}')}[k(\mathbf{z},\mathbf{z}')],\end{aligned} \tag{4.33}$$

where $k(\mathbf{x},\mathbf{y})$ is any positive definite kernel. Disentangled-DiffVAE uses the Gaussian kernel, $k(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2}}$.

Let $\mathbf{X}^B = \{\mathbf{x}^{(i)}\}_{i=1}^B$ be a mini-batch $\mathcal{B}$ with B samples randomly drawn from the full dataset such that $\mathbf{x}^{(i)} \sim p_{\text{data}}(\mathbf{x})$. Let $\mathbf{Z} = \{\mathbf{z}^{(i)}\}_{i=1}^B$ be the sampled latent code computed for each example in $\mathcal{B}$ and let $\mathbf{Z}_\star = \{\mathbf{z}_\star^{(i)}\}_{i=1}^B$ consist of true samples from the prior $\mathbf{z}_\star^{(i)} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$. During training the MMD is computed by comparing true samples in $\mathbf{Z}_\star$ with the samples computed by the encoder $\mathbf{Z}$ over mini-batches of training examples $\mathbf{X}^B$.

Thus, the training objective maximised by Disentangled-DiffVAE is:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{X}^B) = \frac{1}{B}\sum_{\mathbf{x}\in\mathbf{X}^B}\left(\sum_{i=1}^{n} x_i \log x_i' + (1-x_i)\log(1-x_i')\right) \\ + \left(\sum_{i=1}^{B}\sum_{i'=1}^{B} k(\mathbf{z}^{(i)},\mathbf{z}^{(i')}) + \sum_{j=1}^{B}\sum_{j'=1}^{B} k(\mathbf{z}_\star^{(j)},\mathbf{z}_\star^{(j')}) - 2\cdot\sum_{i=1}^{B}\sum_{j=1}^{B} k(\mathbf{z}^{(i)},\mathbf{z}_\star^{(j)})\right).\end{aligned} \tag{4.34}$$

The architecture and hyperparameters used for Disentangled-DiffVAE match the ones of DiffVAE as described in Section 4.2.4. This will allow us to make a comparative evaluation of the two models.

4.4 Graph-DiffVAE

This section explores how graph convolutional networks can be applied to gene expression data. The Graph-DiffVAE model architecture presented, is still part of the probabilistic framework of the variational autoencoder. However, instead of just modelling the stochastic behaviour of gene expression across cell types, Graph-DiffVAE can also model the relations between different cell types.

In this context, we will consider the different cells in a dataset as nodes in a graph. The gene expression measurements for each cell will form the node features. Edges between cells can have different meanings based on the data analysed. For example, section 5.5 will explore a method for building an initial graph for cells in a dataset where an edge encodes similarity between cells (as measured by the Pearson correlation coefficient). Another option is to incorporate biological knowledge in the graph, where for instance, edges can represent potential differentiation trajectories for the cells in a dataset.

4.4.1 Graph convolutional networks

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $|\mathcal{V}| = N$ be an undirected and unweighted initial graph built from the cells, defined by the binary adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$. Such an initial graph for the cells can be already available or it can be artificially built using the methods described in Section 5.5.

Let $\mathbf{X}$ be an $N \times F$ matrix consisting of node features, where F is the number of features for each node. In our case, the nodes are the different cells in the dataset, and the features are represented by the gene expression for each cell. Assume that each node is connected to itself, so that $\mathbf{A}$ has diagonal entries $A_{ii} = 1$. Let $\mathbf{D}$ be the diagonal degree matrix of $\mathbf{A}$: $D_{ii} = \sum_j A_{ij}$.

Graph convolutional networks (GCN) were proposed by Kipf and Welling [6] with the following layer wise propagation rule:

$$\mathbf{X}^{(l+1)} = \tau(\tilde{\mathbf{A}}\mathbf{X}^{(l)}\mathbf{W}^{(l)}) = \mathrm{GCN}_{\tau,l}(\mathbf{A}, \mathbf{H}^{(l)}), \tag{4.35}$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, τ is the activation function applied, $\mathbf{X}^{(l)}$ and $\mathbf{X}^{(l+1)}$ are the activations of the layers (l) and $(l+1)$ respectively, and $\mathbf{W}^{(l)}$ are the weights. $\mathbf{X}^{(0)}$ represents the input feature matrix $\mathbf{X}$. The size of layer n_l, represents by the number of node features computed at layer (l).

Through the layer-wise propagation rule, the graph convolutional network performs spectral graph convolutions. The model can be regarded as the differentiable and generalised version of the algorithm proposed by Weisfeiler-Lehman on graphs. In particular, the layer wise propagation rule in Equation 4.35 can be viewed as a message passing computation over the graph structure. Through one hidden layer, nodes in the graph pass information about their local structure to neighbours that are 1-hop away. Based on the information received from the neighbours, the nodes update their node features.

4.4.2 Graph-DiffVAE details

Graph-DiffVAE is a graph variational autoencoder where the encoder and the decoder networks are graph convolutional networks applying the layer-wise propagation rule described in Equation 4.35.

Graph-DiffVAE is based on the graph variational autoencoder proposed by Kipf and Welling [7] and on the Graphite model developed by Grover *et al.* [64]. However, unlike in the Graphite mode, Graph-DiffVAE uses the initial adjacency matrix $\mathbf{A}$ in the decoder to predict edges.

Inference model (encoder) : $q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{A}, \mathbf{X})$.

The encoder in Graph-DiffVAE is represented by a graph convolutional network with multiple layers and with Gaussian output. The input to the encoder consists of the matrix with node features $\mathbf{X}$ and of the graph adjacency matrix $\mathbf{A}$. The layers in the encoder network perform the following operations:

$$\mathbf{X}^{(1)}_{\text{enc}} = GCN_{\tau_1,1}(\mathbf{A}, \mathbf{X}), \tag{4.36}$$

$$\boldsymbol{\mu} = GCN_{\tau_2,\mu}(\mathbf{A}, \mathbf{X}^{(1)}_{\text{enc}}), \tag{4.37}$$

$$\log \boldsymbol{\sigma}^2 = GCN_{\tau_2,\sigma}(\mathbf{A}, \mathbf{X}^{(1)}_{\text{enc}}). \tag{4.38}$$

$$\tag{4.39}$$

where τ_1 is the ReLU activation function and τ_2 is the linear activation function. The number of node features computed in $\mathbf{X}^{(1)}_{\text{enc}}$ is 512 and the number of node features in $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ is 50, thus forming a latent representation $\mathbf{Z}$ where each node has $M = 50$ features.

The encoder represents a factorised multivariate Gaussian distribution, such that:

$$q_{\phi}(\mathbf{Z}|\mathbf{A},\mathbf{X}) = \prod_{i=1}^{N} q_{\phi}(\mathbf{z}_i|\mathbf{X},\mathbf{A}), \tag{4.40}$$

$$q_{\phi}(\mathbf{z}_i|\mathbf{X},\mathbf{A}) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)). \tag{4.41}$$

The reparametrisation trick is used again to sample each $\mathbf{z}_i$:

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I}), \tag{4.42}$$

$$\mathbf{z}_i = \boldsymbol{\mu}_i + \boldsymbol{\varepsilon} \odot \boldsymbol{\sigma}_i. \tag{4.43}$$

It is important to notice that the latent representation $\mathbf{Z}$ build by the Graph-DiffVAE encoder contains information from both the graph structure and the node features. $\mathbf{Z}$ encompass the latent representations for each node in the graph.

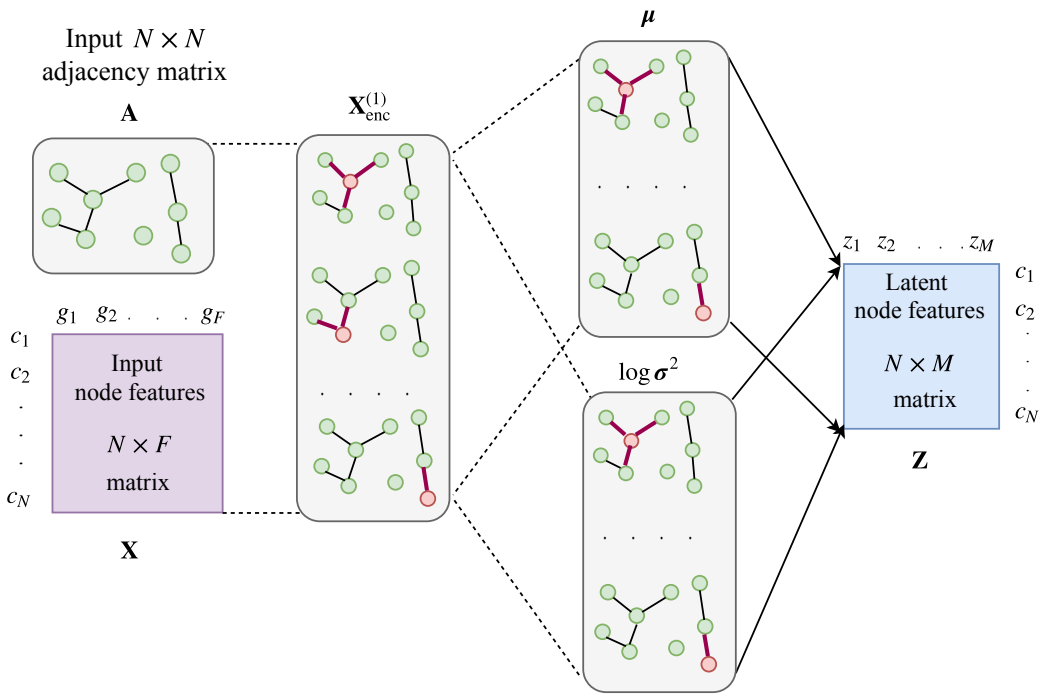


Figure 4.8: Encoder architecture performing graph convolutions: each graph convolutional layer uses the initial graph structure described by $\mathbf{A}$ such that each node in the graph can transmit information to their local neighbours (as illustrated by the red edges). Each layer computes features for the nodes in the graph.

Generative model (decoder): $p_\phi(\mathbf{A}|\mathbf{Z},\mathbf{X})$

The output of the decoder is a adjacency matrix $\hat{\mathbf{A}}$ representing an undirected and unweighted graph with predicted edges between nodes. Such an adjacency matrix can be represented by a factorised Bernoulli distribution.

The decoder network uses as input the initial adjacency matrix $\mathbf{A}$ and a concatenation of th input node features $\mathbf{X}$ and the latent node features computed by the encoder $\mathbf{Z}$, described by $[\mathbf{Z}|\mathbf{X}]$. The layers in the decoder network perform the following operations.

$$\mathbf{X}^{(1)}_{\text{dec}} = GCN_{\tau_1,1}(\mathbf{A}, [\mathbf{Z}|\mathbf{X}]), \tag{4.44}$$

$$\mathbf{Z}' = GCN_{\tau_1,1}(\mathbf{A}, \mathbf{X}^{(1)}_{\text{dec}}), \tag{4.45}$$

$$\mathbf{Z}^\star = \frac{1}{2}(\mathbf{Z}' + \mathbf{Z}), \tag{4.46}$$

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}^\star_i \mathbf{Z}^\star_j). \tag{4.47}$$

where τ_1 is the ReLU activation function. The number of node features computed in $\mathbf{X}^{(1)}_{\text{dec}}$ is 512. The decoder builds its own latent representation $\mathbf{Z}'$ consisting of 50 node features which it then adds to the representation constructed through the encoder to obtain $\mathbf{Z}^\star$.

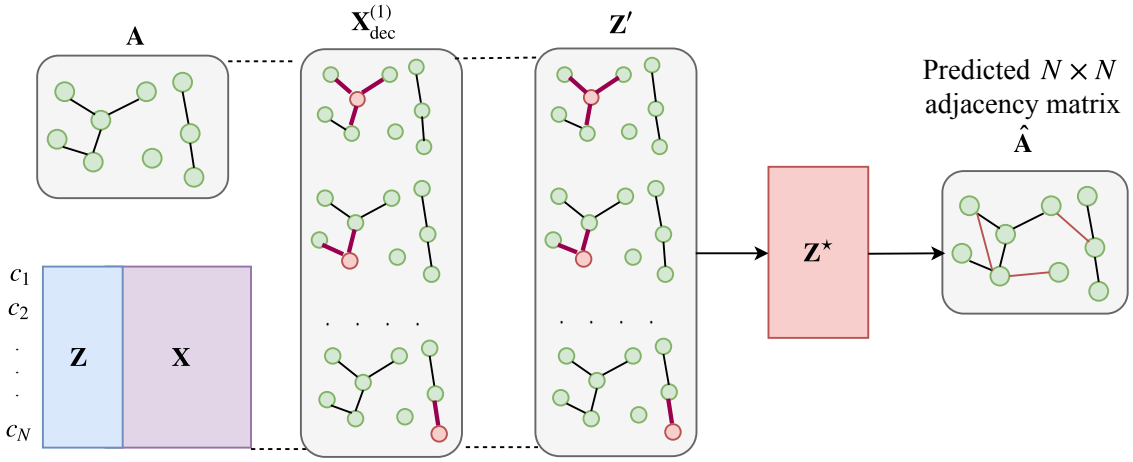


Figure 4.9: Decoder network performing graph convolutions: each graph convolutional layer uses the initial graph structure described by $\mathbf{A}$ so that each node in the graph transmits information to their local neighbours (as illustrated by the red edges). The input is the concatenation of initial node features $\mathbf{X}$ and latent node features computed by the encoder $\mathbf{Z}$, and the output consist of an adjacency matrix.

Training objective

The elements in the output adjacency matrix $\hat{\mathbf{A}}$ are assumed to be independents. Moreover, each element in the output adjacency matrix is modelled as $\hat{A}_{ij} = p_{\boldsymbol{\theta}}(A_{ij} = 1|\mathbf{X}, \mathbf{Z})$. Using the same derivations as Equations 4.22 and 4.21 we obtain that the training objective for Graph-DiffVAE is:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}, \mathbf{A}) = \frac{1}{N^2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}\log A'_{ij} + (1 - A_{ij})\log\left(1 - A'_{ij}\right) + \right) \\ \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\sum_{j=1}^{M}(1 + \log(\sigma_{ij})^2 - \mu_{ij}^2 - \sigma_{ij}^2)\right). \tag{4.48}$$

As we shall see in evaluation Section 5.5, the proposed architecture and training objective for Graph-DiffVAE results in additional edges between cells to be predicted in the output adjacency matrix $\hat{\mathbf{A}}$. The predicted relationships between cells are similar to the ones in the initial graph given as input to the model.

4.5 Implementation details

The variational autoencoder models were implemented in Python using the Keras (version 2.1.6) deep learning framework [65] with TensorFlow (version 1.8.0) [66] back-end. Additional libraries were used for data processing and analysis, including SciPy [67] and Scikit-learn [68]. The models DiffVAE, Disentangled-DiffVAE and Graph-DiffVAE were implemented in a modular manner that allows them to be easily extended and used on gene expression datasets.

4.6 Summary

This chapter introduced the variational autoencoder and described the architecture of DiffVAE, the model proposed to represent the stochastic behaviour of gene expression leading to cell differentiation. Then, the training objective of the variational autoencoder was improved to maximise the mutual information between the input data and the latent representation which resulted in Disentangled-DiffVAE, a model achieving better separation of the biological factors influencing cell differentiation.

Finally, graph convolutional networks were employed to build Graph-DiffVAE, a model capable of not only modelling stochasticity in gene expression among cells, but also potential relationships between individual cells.

Chapter 5

Methodology design and evaluation

This chapter presents the methodology developed for modelling cell differentiation using the neural models proposed in this project. We shall see how the latent dimensions in the variational autoencoders can represent the differentiation of various cells, how driver genes for the differentiation process can be identified, how cell states can be changed through modifications to the latent dimension and how links between cells can be predicted.

The methods developed are general and can be applied to study any cell type. In this chapter, the methodology will be evaluated and exemplified by studying the differentiation of blood stem and mature cells in zebrafish using single-cell RNA-seq data.

5.1 sc-RNA-seq zebrafish data

The unsupervised neural models and the methodology developed in this project are used to analyse single-cell gene expression data characterising cells in zebrafish. The data was obtained through collaborators at the Wellcome Trust Sanger Institute.

The dataset analysed consists of $k = 1845$ gene expression measurements from $N = 1422$ cells: $\mathcal{D} = \{\mathbf{c}^{(i)}\}_{i=1}^{N}$, where $\mathbf{c}^{(i)} = [g_1^{(i)}\ g_2^{(i)}\ \dots g_k^{(i)}]^T$. Athanasiadis *et al.* [69] computationally reconstructed the differentiation trajectories in vitro and five cell

states were found in the dataset using the Monocle2 algorithm [14]: Monocytes, Neutrophils, Erythrocytes, Thromobocytes and HSPCs (Hematopoietic Stem and Progenitor Cells). Figure 5.1 illustrates the differentiation trajectory reconstructed for these cells using Monocle2.

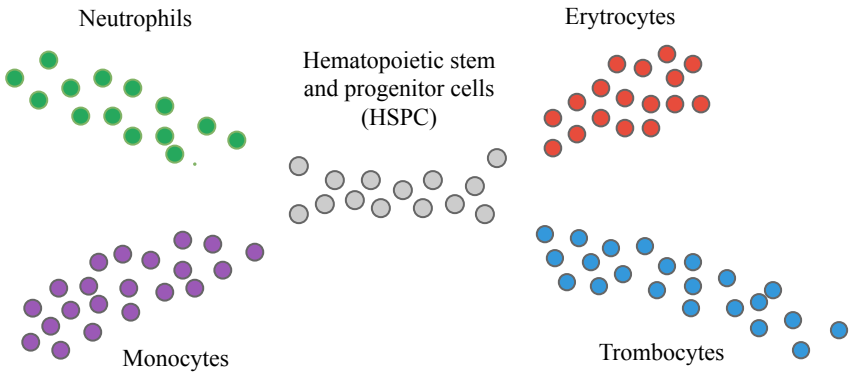


Figure 5.1: Cell states in the sc-RNA-seq zebrafish dataset. The Neutrophils, Erythrocites, Monocytes and Trombocytes are mature blood cells, while the HSPCs consist of stem cell and progenitor cells which are less specialised cells and can differentiate further into the mature blood cells.

The computational framework developed this chapter is based on the unsupervised neural methods proposed in Chapter 4 and it is successfully applied to this sc-RNA-seq zebrafish dataset. In the following sections, we shall see that DiffVAE and Disentangled-DiffVAE can encode the differentiation of the mature blood cells in their latent dimensions (Section 5.2) and subsequently reveal driver genes for the differentiation of Neutrophils, Erythrocites, Monocytes and Trombocytes (Section 5.3). Additionally, Section 5.4 will show how HSPCs can be changed into Neutrophils through operations on the latent dimensions encoding the differentiation of Neutrophils.

An interesting aspect about the dataset is the fact that among the HSPCs, there are some cells that are pure stem cells and some others which are already in the process of differentiation into mature cells. However, labels for such cells do not exist which makes the problem interesting from an unsupervised learning perspective. The links predicted by Graph-DiffVAE (Section 5.5) between the HSPCs and the mature blood cells could be interpreted as potential differentiation trajectories for the less specialised cells.

5.1.1 Data pre-processing

Data pre-processing and in particular normalisation represents a crucial first step in any machine learning pipeline. Thus, the gene expression data was normalised using Min-Max scaling. More specifically, the expression values for the genes, which represent the input features to the models were scaled to the range $[0, 1]$. Let $\mathbf{g}_i = [g_i^{(1)}\ g_i^{(2)}\ \dots\ g_i^{(N)}]$ be the expression values for gene i across the different cells. The following transformation was applied to each component $g_i^{(j)}$ of $\mathbf{g}_i$:

$$g_i^{(j)} = \frac{g_i^{(j)} - \min(\mathbf{g}_i)}{\max(\mathbf{g}_i) - \min(\mathbf{g}_i)}. \tag{5.1}$$

The motivation for choosing this data normalisation is to be able to model the gene expression for each cell as a multivariate Bernoulli distribution in the variational autoencoder models.

5.2 Latent dimensions encoding cell differentiation

DiffVAE and Disentangled-DiffVAE were design to model the data generating process giving rise to the observations in out dataset $\mathcal{D}$. Thus, these methods should be able to identify the biological mechanisms that result in the observed gene expression value for our cells.

To understand what the latent dimensions in each model have learnt about the cells, an initial experiment was performed where latent dimensions from both models were plotted against each other, as illustrated in Figure 5.2. The results show that, in DiffVAE, the predicted value in latent dimension 22 for the Thrombocytes is higher than for the other cells. A similar outcome is observed for Disentangled-DiffVAE, where latent dimension 28 distinguishes Monocytes.

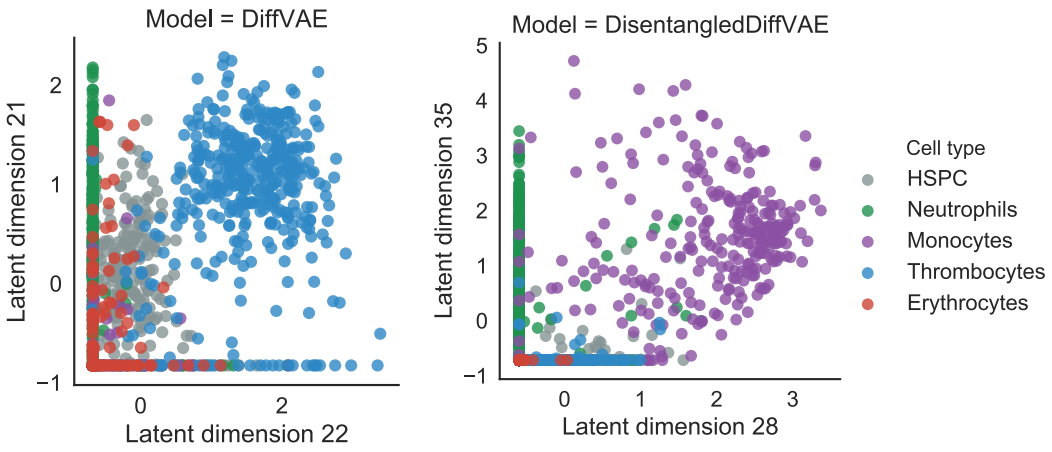


Figure 5.2: Latent dimensions in DiffVAE (*left*) and Disentangled-DiffVAE (*right*) plotted against each other. This visualisation indicates the fact the models can encode the differentiation of particular types of cells in the latent dimensions.

This initial empirical analysis was helpful for giving us insight into the models' capabilities for encoding cell differentiation. However, to perform a proper evaluation, a more rigorous methodology needed was developed.

Consider the analysis of a latent dimension k for any of the models. Let $\mathbf{z}_k = [z_k^{(1)}\ z_k^{(2)}\ \dots z_k^{(N)}]^T$ be the predicted value of the encoder for z_k across all of the cells in the dataset. Let μ_k and σ_k be the mean and standard deviation of $\mathbf{z}_k$. We define

$$\mathcal{D}_k = \{\mathbf{c}_i \in \mathcal{D} | z_k^{(i)} \geq \mu_k + \sigma \vee z_k^{(i)} \leq \mu_k - \sigma\} \tag{5.2}$$

as the set of cells at least a standard deviation from the mean in latent dimension k. By computing the percentage distribution of the cells in $\mathcal{D}_k$ across the distinct cell types found in the dataset, we can evaluate how well the latent dimension is encoding the differentiation of a particular type of cell.

Figure 5.3 gives an illustration of the analysis performed on each latent dimension to obtain this percentage distribution across cell types.

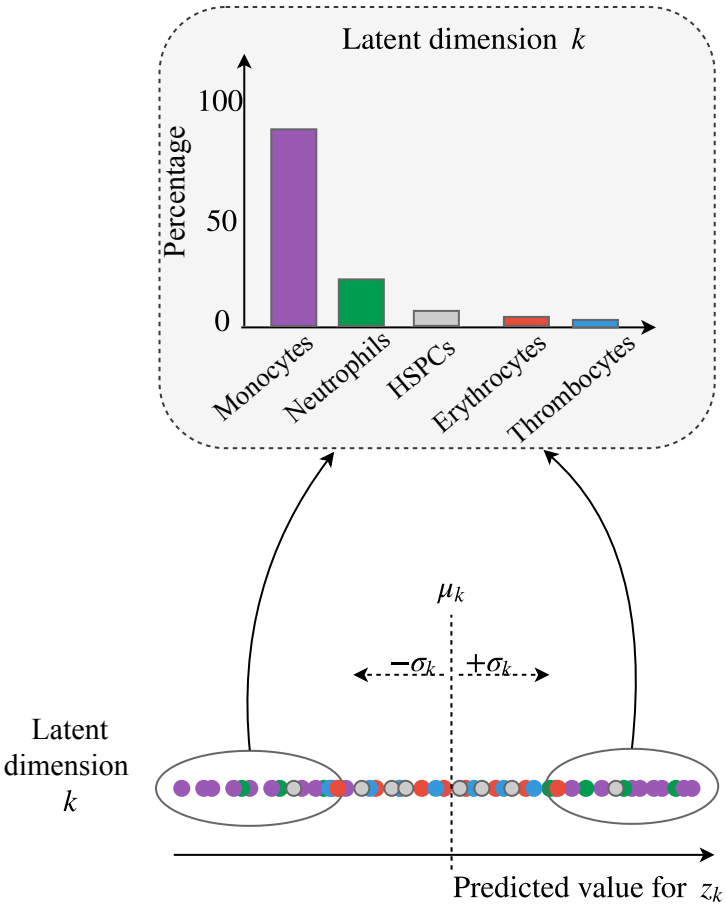


Figure 5.3: Description of the analysis performed on each latent dimension k to study its capacity of encoding the differentiation of each type of cell. The percentage distribution of cells for which $z_k \geq \mu_k + \sigma_k \vee z_k \leq \mu_k - \sigma_k$ was computed across the cell types. The histogram at the top of the figure represents an example of this percentage distribution for latent dimension k.

This methodology was evaluated on the latent dimensions in DiffVAE and Disentangled-DiffVAE and Figures 5.4 and 5.5 illustrate the percentage distributions obtained. For further analysis, we will say that a latent dimension k is able to differentiate a cell X when the percentage of cells of type X in $\mathcal{D}_k$ is larger than 75%.

The results obtained indicate that both DiffVAE and Disentangled-DiffVAE have several latent dimensions capable of differentiating Monocytes, Neutrophils and Thrombocytes.

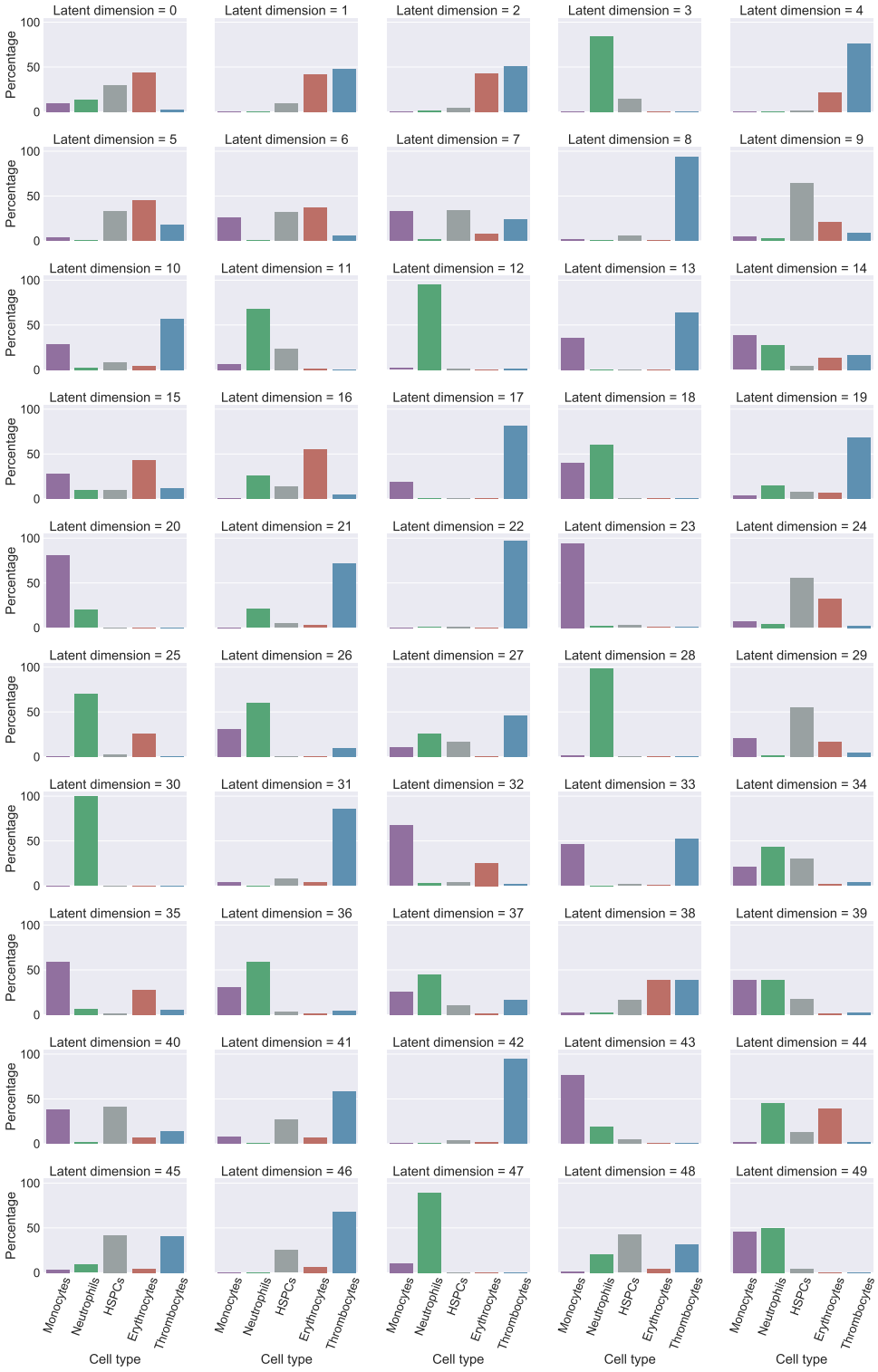


Figure 5.4: Percentage distributions of latent dimensions in DiffVAE. See Figure 5.3 for description of the method used to compute them.

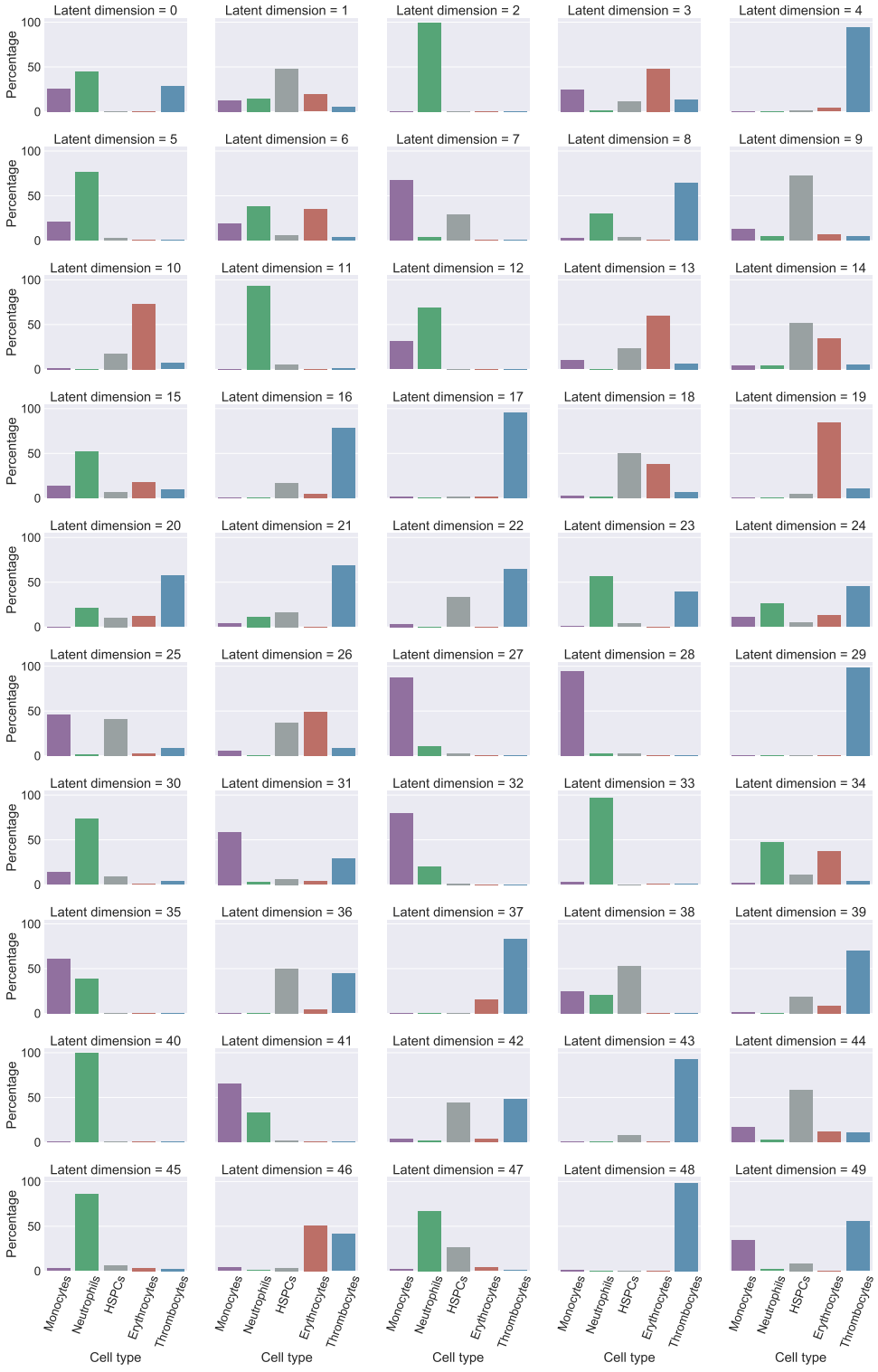


Figure 5.5: Percentage distributions of latent dimensions in Disentangled-DiffVAE. See Figure 5.3 for description of the method used to compute them.

However, it is noticeable that for Disentangled-DiffVAE more latent dimensions are able to differentiate the mature blood cell type. This result is expected given the fact that we have changed the loss function in order to maximise the mutual information between the input and the latent dimension and to disentangle the factors of variation in the data more effectively. The superiority of Disentangled-DiffVAE is also enhanced by the fact that some of its latent dimensions (e.g latent dimension 19) can differentiate Erythrocytes. None of the latent dimensions of DiffVAE can achieve this.

One important aspect that was noticed experimentally is that the latent dimensions differentiating particular types of cells were not necessarily the dimensions with greatest variance. Thus, the methods described in this project can reveal more information about what is encoded in the latent dimensions rather than standard methods what are usually based on measures of variability.

5.3 Identifying high weight genes

The latent dimensions of the variational autoencoder was designed to capture the important biological processes that have led to cell differentiation. The decoder learns to reconstruct the original gene expression data, and therefore, the weights in the decoder indicate the contribution of each gene in the biological process. Figure 5.6 illustrates the idea behind the methodology described in this section.

As we have already seen, the latent dimensions in DiffVAE and Disentangled-DiffVAE can encode the differentiation of cells. By finding the high weight connections between theses latent dimension and the reconstructed gene expression, we can identify the driver genes for the cell differentiation process.

These high weight connections are identified using the weight matrices in the decoder. The decoder consists of a multiple fully connected layers. Let $\mathbf{z} \in \mathbb{R}^m, \mathbf{h}^{(1)} \in \mathbb{R}^{n_1}, \dots \mathbf{h}^{(l)} \in \mathbb{R}^{n_l}, \mathbf{g} \in R^n$, be the sequence of layer activations in the decoder, where the latent dimension $\mathbf{z}$ represents the input, $\mathbf{h}^{(1)} \dots \mathbf{h}^{(l)}$ are the hidden layers and $\mathbf{g}$ is the output.

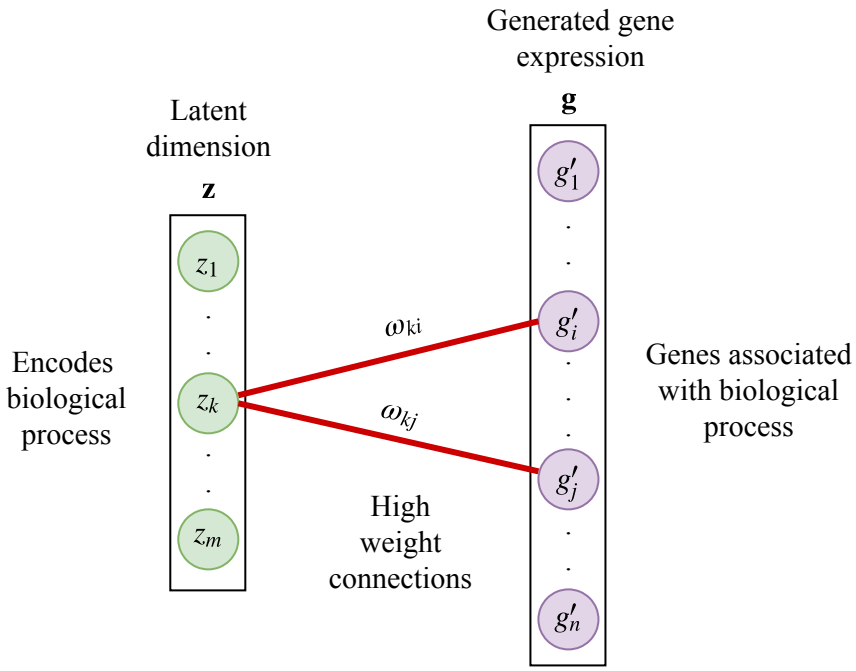


Figure 5.6: Finding the high weight connections between the latent dimension and the reconstructed output can help in identifying genes associated with a biological process.

The weight matrices for the connections between the layers in the decoder can be described by $W^{(0)} \in \mathbb{R}^{m \times n_1}, W^{(1)} \in \mathbb{R}^{n_1 \times n_2}, \ldots, W^{(l)} \in \mathbb{R}^{n_l \times n}$. Let $\boldsymbol{\omega} \in \mathbb{R}^{m \times n}$ be the weight matrix for the connections between the latent dimension and the output. $\mathbf{w}$ can be computed by multiplying the weight matrices between the individual fully connected layers, as follows:

$$\boldsymbol{\omega} = \mathbf{W}^{(0)} \cdot \mathbf{W}^{(1)} \cdot \ldots \cdot \mathbf{W}^{(l)}, \tag{5.3}$$

where the matrix element ω_{ij} indicates the weight of the connection between latent dimension i and gene j.

For each latent dimension, the genes are sorted by the absolute value of their weight. The genes having the highest of such weights are referred to the high weight genes.

5.3.1 Genes influencing cell differentiation

Additional evaluation was performed by assessing the capabilities of DiffVAE and Disentangled-DiffVAE to find the driver genes for the differentiation process. For this, we used the information obtained about each model regarding their latent dimension's capability to indicate the differentiation of a specific type of cells (Figures 5.4 and 5.5).

For each model, where possible, we selected the latent dimension that distinguished the best Monocytes, Neutrophils, Erythrocytes and Thrombocytes and computed the high weight genes, i.e. the genes with the high weight connections to the latent dimension. Table 5.1 summarises the results and also provides references in literature that validate the findings of the autoencoder.

Cell type	High weight genes obtained from DiffVAE	High weight genes obtained from DisentangledDiffVAE
Monocytes	Latent dim 23: c1qa [69], grna, lgals2a, slc3a2a, mafbb [70, 71]	Latent dimension 28: lgals2a, c1qc, c1qa [69], gals3bpb, s100a10b, mafbb [70, 71]
Neutrophils	Latent dim 30: cfl1l, scpp8, lect2l, ponzr6, mpx [69, 72], npsn [72]	Latent dimension 2: illr4 [69], ponzr6, npsn [72], abcb9 [73], cfl1l, lyz [74]
Erythrocytes	None of the latent dimensions are reliably differentiating erythrocytes.	Latent dimension 19: alas2 [69], ba1l [69], aqp1a.1 [69], hbaa1 [69], slc4a1a [75, 76], ba1 [69]
Thrombocytes	Latent dimension 42: ctgfa, fn1b [69], tspan7, tuba8l3, thbs1b, cldn5b, bmp16	Latent dimension 29: fn1b [69], itga2b [69, 77], bmp6, thbs1b [69], fhl1a, ctgfa, apln

Table 5.1: High weight genes computed using the high weight connections to the latent dimensions with the highest percentage for differentiating the corresponding cell type. Results are validated by references to scientific literature confirming the findings of the DiffVAE and Disentangled-DiffVAE.

5.4 Navigating cell types

Another methodology developed in this project involves changing the state of cells through operations on the latent dimension. This can help us learn more about the the type of biological changes in gene expression that cause a less specialised cell such as an HSPC to differentiate in a more specialised cell such as a Neutrophil.

Assume that we have identified, using the methods described in Section 5.2 that latent dimension j encodes the differentiation of a type of mature blood cells, such as Neutrophils. Let μ_j and σ_j be the mean and standard deviation of $\mathbf{z}_j = [z_j^{(1)}\ z_j^{(2)}\ \dots z_j^{(N)}]^T$ the predicted value of the encoder for z_j across all of the cells in the dataset.

Using the definitions in Section 5.2, we can say that if latent dimension j differentiates Neutrophils, it means that the ratio of the number of Neutrophils in $\mathcal{D}_j = \{\mathbf{c}_i \in \mathcal{D} | z_j^{(i)} \geq \mu_j + \sigma_j \vee z_j^{(i)} \leq \mu_j - \sigma_j\}$ is larger than 75%. This strongly suggests that shifting $z_j^{(lk)}$ by the standard deviation σ_j of latent dimension j could potentially change cell $\mathbf{c}_k$ into a Neutrophil. However, as we shall notice in the next section, changing just one latent dimension by one standard deviation is not enough.

The method proposed for changing a less specialised cell (an HSPC) into Neutrophil involves shifting several of the latent dimensions encoding the differentiation of Neutrophils proportionally with their standard deviations. The proportionality factor is the parameter λ. The method is illustrated in Figure 5.7 and it can be used to obtain any of the mature blood cells that the latent dimensions of the variational autoencoder can differentiate. The following section will discuss the results obtained by performing such operations on the cell types.

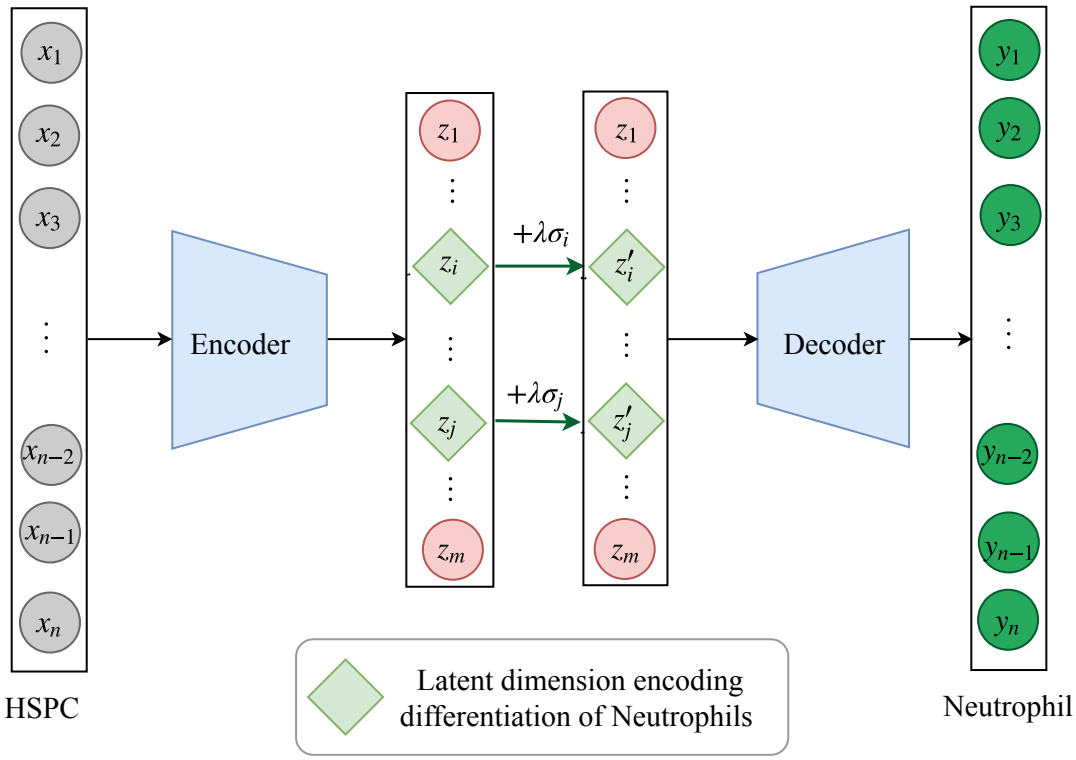


Figure 5.7: Methodology proposed for changing the cellular states: an HSPC can be converted into a Neutrophil by shifting the latent dimensions differentiating Neutorphils by proportional with their standard deviation (as specified by the proportionality factor λ).

5.4.1 Results from changing cell types

To distinguish between the type of cells, we trained a neural network classifier capable of labelling Monocytes, Neutrophils, Erythrocytes and Thrombocytes using the full gene expression data with 99.5% accuracy. The same type of model was used to compare the methods on performing dimensionality reduction, and its details are described in Section 6.4.

As the latent representation in Disentangled-DiffVAE is more informative about the differentiation of the cells in our dataset, we will apply the methodology described in the previous section to this model. The latent dimensions in Disentangled-DiffVAE differentiating Neutrophils are: $2, 5, 11, 40, 45$ (these dimensions can also be inferred from Figure 5.5).

To evaluate the method we performed to following steps: the encoder network was used to compute the latent representation from the gene expression data of HSPCs, then the latent dimensions differentiating Neutrophils were shifted, and then the gene expression data was reconstructed using the decoder. Table 5.2 shows the number of HSPCs that were classified as Neutrophils after shifting the latent dimensions specified.

We can notice that changing just one or two latent dimensions does not change many HSPC into Neutrophils which indicates that the latent dimensions are learning distinct properties of the cell types. Moreover, the number of standard deviations we use to shift the latent dimensions from their mean, as specified by parameter σ is also important in changing HSPCs into Neutrophils.

Number of HSPCs classified as Neutrophils (out of 245)

Shifted latent dimensions (differentiating Neutrophils)	Shifting parameter λ		
	$\lambda = 1$ $(+\sigma)$	$\lambda = 2$ $(+2\sigma)$	$\lambda = 3$ $(+3\sigma)$
Latent dim 2	16	18	22
Latent dim $2, 5$	18	32	44
Latent dim $2, 5, 11$	22	46	78
Latent dim $2, 5, 11, 33$	32	70	117
Latent dim $2, 5, 11, 33, 40$	36	73	127
Latent dim $2, 5, 11, 33, 40, 45$	41	87	152

Table 5.2: Number of HSPCs classified as Neutrophils after incrementally shifting the latent dimensions differentiating Neutrophils in Disentangled-DiffVAE by the number of standard deviations specified by λ.

Further evaluation was performed by also changing the latent dimensions encoding Neutrophils in Disentangled-DiffVAE in the latent representation of the mature blood cells. We then computed the percentage of each type of cell that was classified as a Neutrophils after performing these changes on the latent dimensions. The results are displayed in Table 5.3.

Percentage of each cell type converted to Neutrophils after shifting latent dimensions: $2, 5, 11, 33, 40, 45$			
	Shifting parameter λ		
Cell type	$\lambda = 1$ $(+\sigma)$	$\lambda = 2$ $(+2\sigma)$	$\lambda = 3$ $(+3\sigma)$
HSPC	**16.7**%	**35.8**%	**62.0**%
Monocytes	12.8%	29.2%	49.6%
Thrombocytes	0.4%	0.9%	2.4%
Erythrocytes	11.2%	23.60%	37.5%

Table 5.3: Percentage of each type of cell classified as Neutrophils after shifting the epecified latent dimensions by their standard deviation multiplied by parameter λ.

It is noteworthy that the HSPCs can be most easily changed into Neutrophils. This results are expected and validate our method considering that HSPCs are less specialised cells with more differentiation capabilities.

5.5 Predicting links between cells

This section develops the methodology for using Graph-DiffVAE to understand cell differentiation by predicting links between cells. We will evaluate Graph-DiffVAE by suggesting one method for building the initial graph structure for the cells and by analysing the predicted links. The cells in our dataset are modelled as nodes in a graph where the node features are given by the gene expression levels.

The encoder part of Graph-DiffVAE uses this initial graph structure and the node features to performs graph convolutions, thus predicting latent node representation. Through the graph convolutions, the nodes in the input graph aggregate information from their local neighbours to build more complex node features. The latent node representations computed by the encoder are used in the decoder network, along with the input node features and input graph to predict new links between the cells. The specific mathematical operations performed in the encoder and decoder were described in Section 4.4.

Figure 5.8 illustrates framework which uses Graph-DiffVAE to predict connections between cells. From a methodological perspective, Graph-DiffVAE could help us explore intermediate cell states in the differentiation process. For example, out of the HSPCs the cells that are already in the process of differentiation could be linked to the mature blood cells they could become.

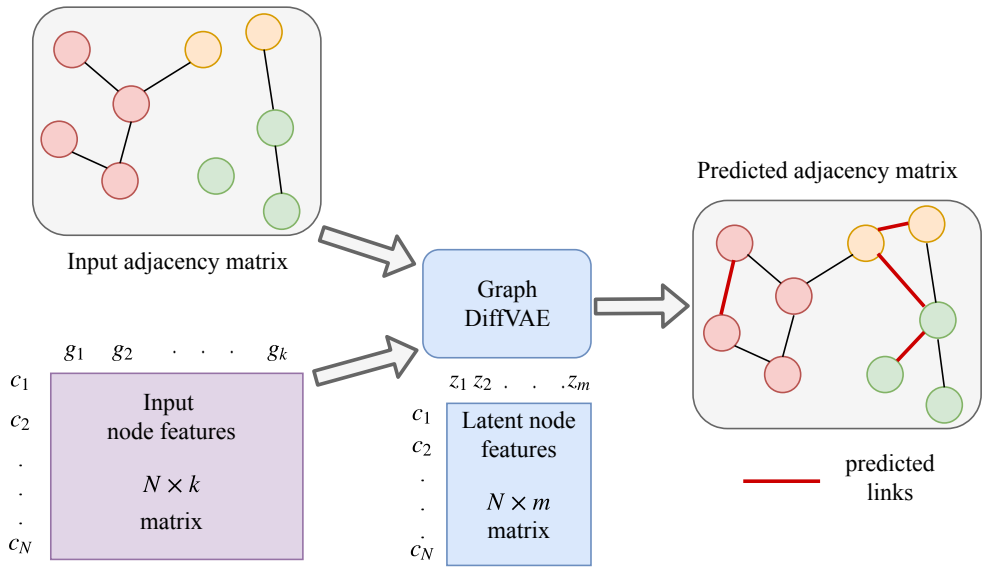


Figure 5.8: Methodology for using Graph-DiffVAE to predict links between cells. Model uses the adjacency matrix for the initial graph and a set of node features to compute a latent representation for each node through the encoder and to predict new links in the graph through the decoder.

5.5.1 Building initial graph for cells

To evaluate Graph-DiffVAE we propose building an initial graph for the cells where there is an edge between each cell and the cell most similar to it. For this purpose, we will use the Pearson correlation coefficient (PCC), which represents an appropriate measure of similarity.

The Pearson correlation coefficient computes similarity between two feature vectors: $\mathbf{x} = [x_1\ x_2\ \ldots\ x_m]^T$ and $\mathbf{y} = [y_1\ y_2\ \ldots\ y_m]^T$ as follows:

$$PCC(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^{m}(x_k - \mu_{\mathbf{x}})(y_k - \mu_{\mathbf{y}})}{\sqrt{\sum_{k=1}^{m}(x_k - \mu_{\mathbf{x}})^2 \sum_{k=1}^{m}(y_k - \mu_{\mathbf{y}})^2}}, \tag{5.4}$$

where

$$\mu_{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{and} \qquad \mu_{\mathbf{y}} = \frac{1}{m}\sum_{i=1}^{m} y_i. \tag{5.5}$$

The value computed by $PCC(\mathbf{x}, \mathbf{y})$ is in the range $[-1.0, 1.0]$, where 1.0 means positive correlation, -1.0 negative correlation and 0.0 no correlation.

This initial graph is undirected and is represented by a binary adjacency matrix where 1 indicates that there is an edge between two nodes (cells). For each cell in the dataset, we computed the Pearson correlation coefficient between its gene expression vector and the feature vectors of the rest of the cells in the dataset and we added an edge to connect it to the highest positively correlated cell.

Figure 5.9 shows these initial connections between the cells. The 2-dimensional coordinates of the nodes were obtained by computing the t-Distributed Stochastic Neighbor Embedding (TSNE) on the latent features learnt by the encoder. The algorithm for TSNE is described in details in Section 6.2.1.

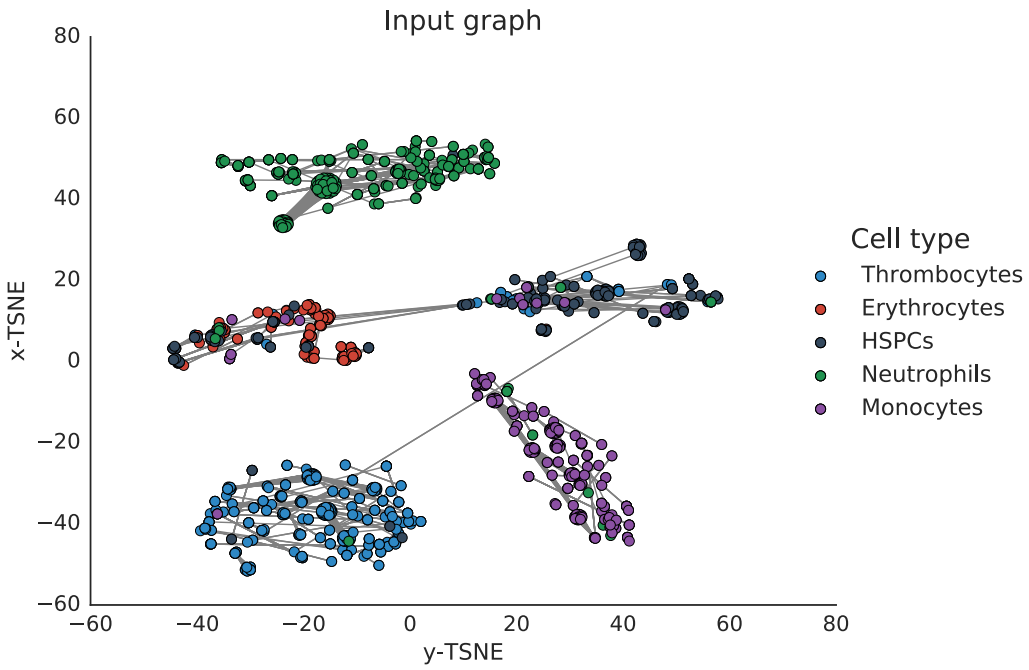


Figure 5.9: Initial graph used as input to Graph-DiffVAE. The position of the nodes (cells) is given by the t-SNE embedding of the latent features computed by the encoder, which indicates the clustering latent representation built by the model.

It is noticeable that the latent representation built in the encoder exhibits a clustering structure between the different types of cells. This is expected and validates the behaviour of the model, as the cells that are highly correlated to each other are more likely to be part of the same cluster.

The predicted links between the cells are illustrated by the adjacency matrix of the output graph in Figure 5.10. As we can see, this clustering behaviour represented in the encoder is emphasised in the output of the decoder, which predicts relatively well defined clusters for the Monocytes, Neutrophils, Erythrocites and Thrombocytes.

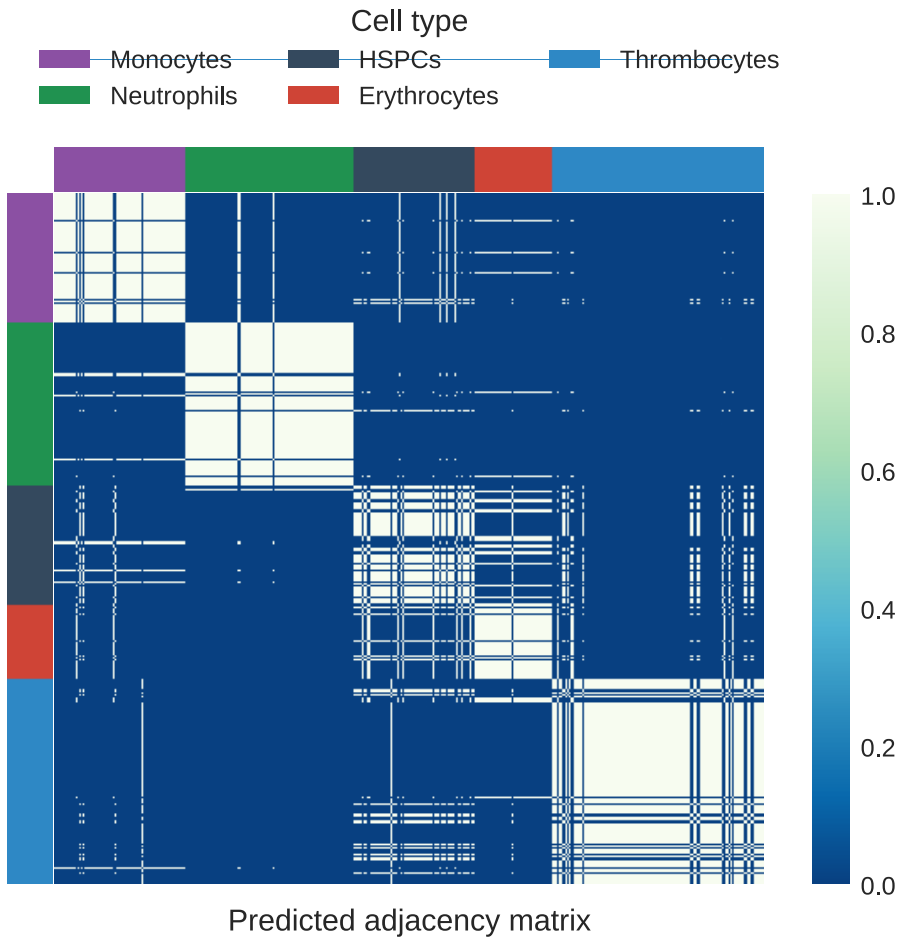


Figure 5.10: Adjacency matrix predicted by Graph-DiffVAE. The white regions indicate the edges predicted by the model with high probability.

An interesting aspect of the predicted graph is the fact that the HSPCs do not cluster together that well. In particular there are clear links between several

HSPCs and all of the other cells in a cluster of mature blood cells. As mentioned in Section 5.1, among the HSPCs there are cells that have already started the process of differentiation towards one of the specific mature cells. The links predicted by Graph-DiffVAE could indicate this aspect.

Further analysis was done by inspecting the specific links between HSPCs and Monocytes and Neuthorphils. Figure 5.11 shows the predicted adjacency matrix consisting only of edges between the HSPCs and Monocytes and Neutrophils. For visualisation purposes, the edges between the same type of cells were removed.

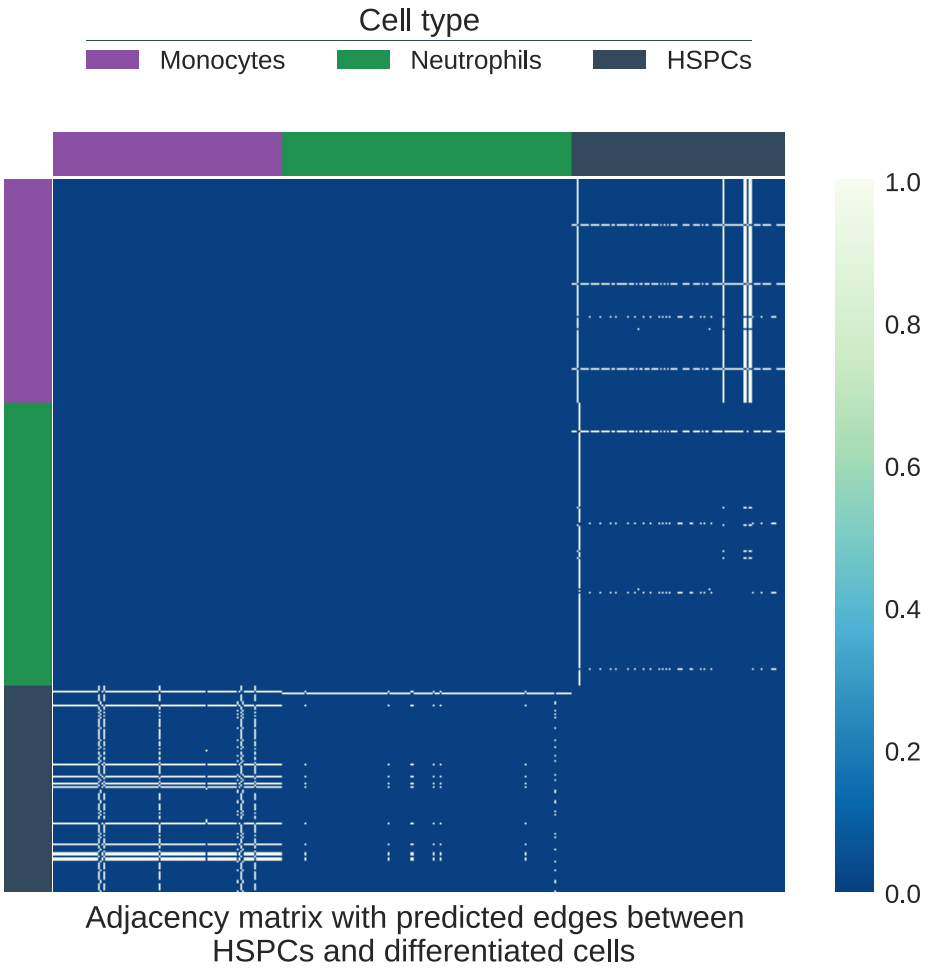


Figure 5.11: Predicted adjacency matrix with links only between HSPCs, Monocytes and Thrombocytes. The white regions indicate the edges predicted by Graph-DiffVAE with high probability.

In addition, for a better comparison between the initial and predicted graph we illustrated in Figure 5.12 both the input edges between the HSPCs and mature blood cells as well as the predicted ones. We can notice that having an initial edge between these types of cells encourages the prediction of similar types of edges.

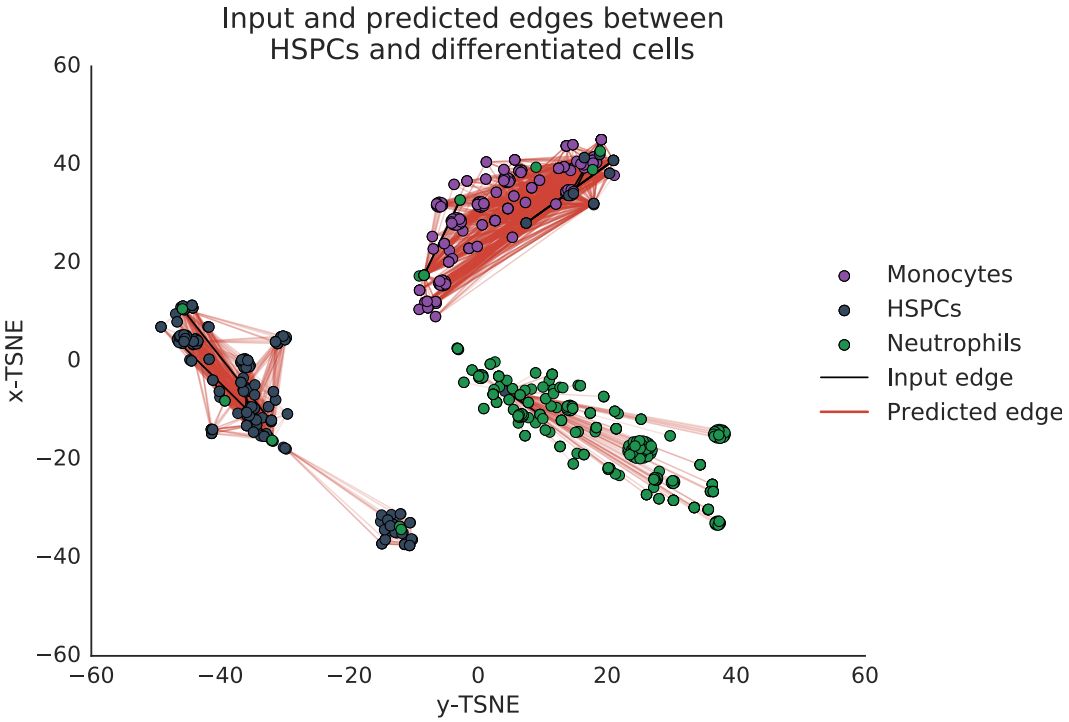


Figure 5.12: Visualisation of input (black) and predicted edges (red) between HSPCs and mature blood cells, namely Monocytes and Thrombocytes.

5.6 Summary

In this chapter, we developed the computational framework for analysing the data representations built by the unsupervised neural models. Firstly, the chapter presented the biological dataset used to apply the methodology. Then, through the methods proposed, we were able to identify latent dimensions in DiffVAE and Disentangled-DiffVAE distinguishing the mature blood cells and the driver genes for the cell differentiation process. Moreover, by shifting latent dimensions, the gene expression of HSPCs was changed enough so that they would be classified as Neutrophils. Lastly, the Graph-DiffVAE model was able to predict edges between correlated (similar) cells.

Chapter 6

Evaluation of dimensionality reduction methods

As sc-RNA sequencing methods resuls in datasets with a large number of gene expression measurements per data point, dimensionality reduction techniques are needed before analysing this data. In this chapter, the suitability of DiffVAE and Disentangled-Diff VAE for performing dimensionality reduction will be assessed through a comparison with other common dimensionality reduction techniques.

6.1 Baseline models description

The baseline models used for comparison on performing dimensionality reduction are Principal Component Analysis (PCA) and a simple autoencoder (SimpleAE), as the one described in Section 2.3.

PCA: The most commonly used linear dimensionality reduction technique, PCA performs an orthogonal transformation on the n-dimensional data and projects it into a coordinate system with m orthogonal axes, where $m < n$. The principal components are the eigenvectors of the data covariance matrix corresponding to the highest eigenvalues. This way, the first principle component will be the direction in which the dataset has the highest variance; the subsequent principal components will be constrained to be orthogonal to the previous ones while maintaining the highest possible variance in the dataset.

SimpleAE: Neural model performing non-linear dimensionality reduction by applying a series of fully connected layer to the n-dimensional data to obtain an m-dimensional representation, where $m < n$. The architecture of SimpleAE matches the one of DiffVAE. Table 6.1 summarises the hyperparameter setting used to train SimpleAE. The loss minimised during training is the binary cross-entropy between the autoencoder input $\mathbf{x}$ and reconstructed output $\mathbf{x}'$:

$$\mathcal{L}_{\text{AE}}(\boldsymbol{\theta}; \mathbf{x}) = -\sum_{i=1}^{n} x_i \log x'_i + (1 - x_i) \log (1 - x'_i), \tag{6.1}$$

where $\boldsymbol{\theta}$ are the parameters in the autoencoder network.

Hyper-parameter	Value setting
Encoder layers	FC(512) - ReLU activation, FC(256) - ReLU activation
Decoder layers	FC(256) - ReLU activation, FC(512) - ReLU activation
Learning rate	0.001
Batch size	256
Training epochs	50
Optimiser used	Adam

Table 6.1: Hyperparameter setting for SimpleAE. Fully connected (FC) layers are described by their size, i.e FC(512) indicates a fully connected layer consisting of 512 neurons.

The performance of the models on dimensionality reduction was assessed for various setting of the reduced dimension size (specified by parameter m) through the following methods:

- manifold embeddings will be applied to the latent representation to analyse how well each model can disentangle the five types of cells in the dataset;
- clustering of both the latent representation and of the manifold embedding for the latent representations will be performed;

- models will be compared on how well the latent representation can be used in classification tasks.

6.2 Manifold embeddings

Manifold embeddings are commonly used for visualizing the representation produced by dimensionality reduction algorithms in a 2-dimensional space [78]. This section explore t-Distributed Stochastic Neighbour Embedding (t-SNE) [79] and Spectral Embeddings [80] for the obtaining 2-dimensional coordinates for the cells, thus allowing us to analyse how well the reduced dimension can disentangle the cell types.

Note that these two methods are not reversible, in the sense that original representation cannot be reconstructed from the 2-dimensional embedding. Moreover, manifold embeddings are not generally applied to more than 50 dimensions. Thus, they can be used for producing visualisations, but not for performing dimensionality reduction in the first instance.

6.2.1 t-SNE

t-SNE assigns each high dimensional data point with a position in a 2-dimensional map through an algorithm consisting of two steps. Firstly, t-SNE builds a probability distribution over the pairs of high dimensional data points, where the probability of choosing similar points is high. Secondly, a similar probability distribution is defined over the points in the 2-dimensional map. Then, the t-SNE algorithm minimises the Kullback-Leibler divergence between the joint probabilities of the high-dimensional data and low-dimensional embedding using a non-convex cost function. t-SNE embeddings are very commonly used in visualizing gene expression data [81].

Figure 6.1 illustrates the results obtained when performing t-SNE on the reduced dimensions computed by the models for $m = 50$. A comparison of the models indicates that the t-SNE of the representation built through the Disentangled-DiffVAE can best separate the cell types.

Conversely, the t-SNE of PCA and of SimpleAE separates the Erythrocytes into three clusters, which is not desirable. Moreover, the embedding of PCA does not give a clear delimitation between Thrombocytes and HSPCs.

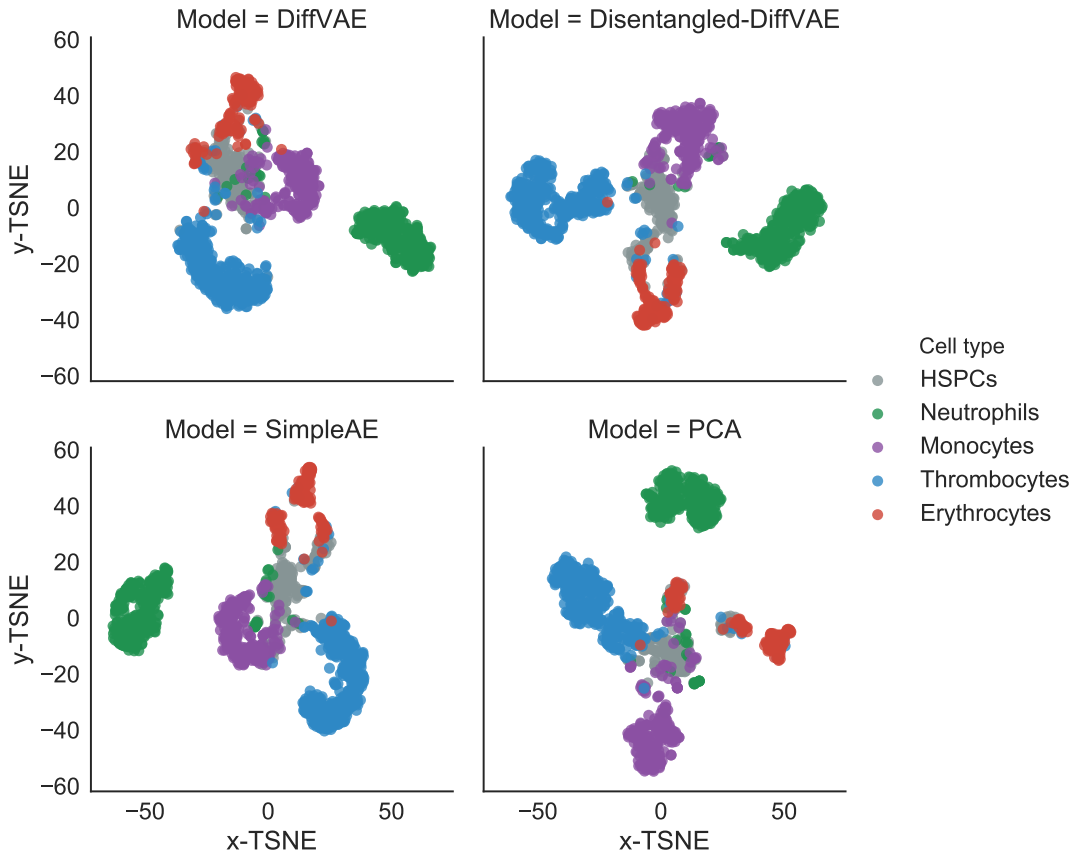


Figure 6.1: t-SNE embeddings of the reduced dimensions ($m = 50$) obtained using the four models compared. The different colours represent the five cell types, as described in the legend on the right.

6.2.2 Spectral embedding

Spectral embedding constructs an adjacency matrix from the high-dimensional data based on the k-Nearest Neighbours. Then, the algorithm performs eigenvalue decomposition on the graph Laplacian and maps points close to each other on the high-dimensional manifold to points close to each other in the low-dimensional space.

Figure 6.2 shows the spectral embeddings of the reduced dimensions for $m = 50$ computed by the model. DiffVAE and Disentangled-DiffVAE are the only models having a spectral embedding where three types of cells, namely the Monocytes, Neutrophils and Thrombocytes are highlighted. However, it is noticeable that none of the spectral embeddings of the models can clearly separate the Erythrocytes.

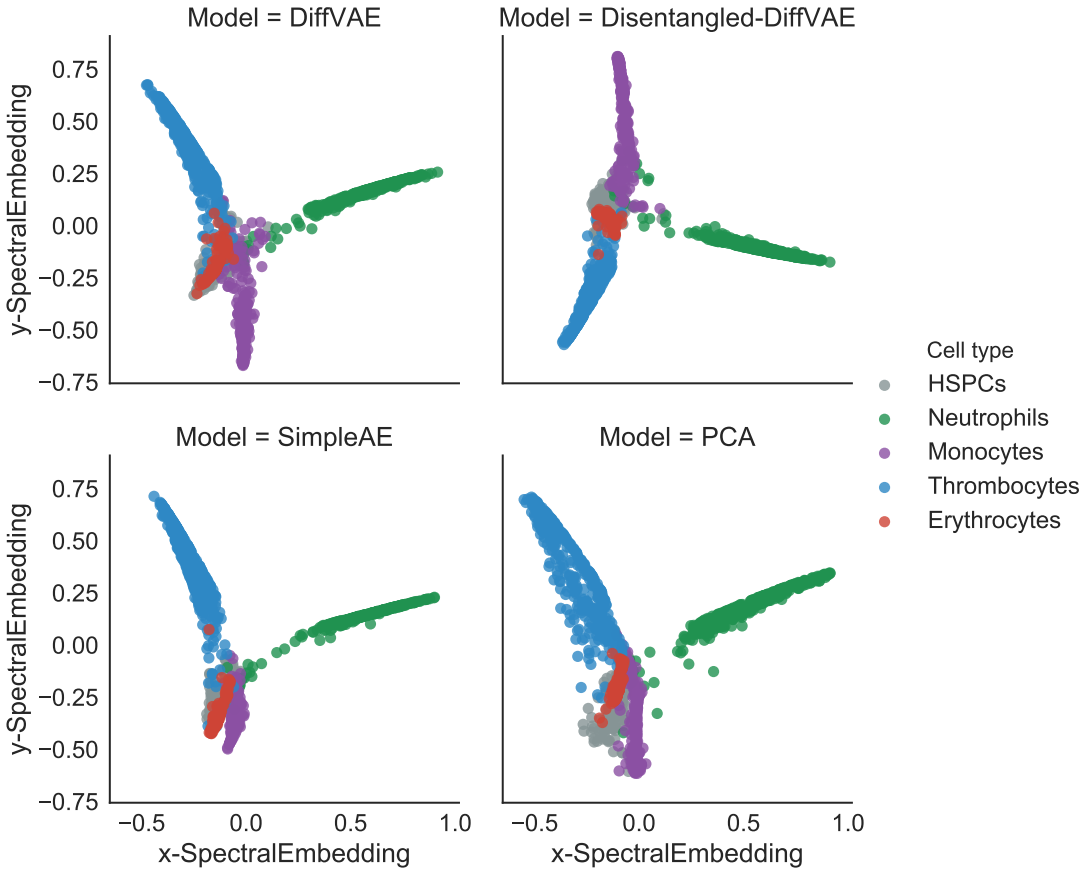


Figure 6.2: Spectral embeddings of the reduced dimensions ($m = 50$) obtained using the four models compared. The different colours represent the five cell types, as described in the legend on the right.

A comparison of the t-SNE and spectral embedding results also highlights the fact that the method chosen for visualizing the reduced has a very important role in helping us asses various dimensionality reduction algorithms.

6.3 Clustering low-dimensional representation

In many situations, the cell types are unknown. As it was also described in the scRNA-seq workflow in section 2.4, clustering is an important step in identifying the cell types. This section evaluates how well each dimensionality reduction technique can be used to identify the cell types. Their performance will be compared using two common clustering algorithms: k-means and Gaussian mixture modeling (GMM). We will cluster both the raw data obtained through dimensionality reduction, as well as the 2-dimensional embedding produced using t-SNE.

k-means and GMM are partitioning clustering algorithms that divide the input data points into a specified number of clusters and adjust the membership of the data points to each cluster in a iterative manner. The main difference between the two methods is that k-means performs hard clustering (one data point is assigned to a single cluster), while GMM is a probabilistic model that performs soft clustering (each data point has a probability of being part of each cluster).

The clustering algorithms are evaluated using the Adjusted Rand Index (ARI) score which is a good measure of similarity between the clustering produced by the algorithms and the true cell types [82]. ARI takes values in $[0.0, 1.0]$, where 0.0 indicates random clustering predictions, and 1.0 indicates that the predicted and true clusters are the same up to a permutation.

The dimensionality reduction methods were evaluated using two settings for the reduced dimension size. For each setting, the clustering algorithms (including the computation of the t-SNE embedding) were performed 50 times and each time the ARI between the true labels and the cluster labels was computed. The results reported in this section are given by the mean ARI obtained and its standard error.

Table 6.2 consists of the results obtained after clustering the reduced m-dimensional data computed by the models compared. It is noticeable that the Disentangled-DiffVAE performs best when $m = 50$, and the SimpleAE when $m = 30$. An explanation for this could be that while the Disentangled-DiffVAE can best separate the cell types, it needs a larger number of latent dimensions to disentangle the factors of variation in the data. Nevertheless, it is clear that all of the autoencoder models achieve better performance than PCA.

As a reference for evaluation, clustering was also performed on the full set of 1845 genes. The ARI score obtained for k-means was 0.612 ± 0.000, while for GMM it was 0.409 ± 0.023. The results in tables also straighten the importance of a dimensionality reduction step before applying clustering.

Clustering method	Dim size (m)	**DiffVAE**	**Disentangled DiffVAE**	**Simple AE**	**PCA**
K-means	30	0.792 ± 0.000	0.809 ± 0.000	**0.851**± 0.000	0.638 ± 0.000
	50	0.775 ± 0.000	**0.829** ± 0.000	0.811 ± 0.000	0.629 ± 0.000
GMM	30	0.693 ± 0.004	0.745 ± 0.010	**0.779**± 0.014	0.638 ± 0.012
	50	0.706 ± 0.006	**0.732** ± 0.005	0.653 ± 0.007	0.541 ± 0.013

Table 6.2: Mean ARI obtained for clustering the reduced dimension for two settings of the reduced dimension size m and the standard error in the results.

Table 6.3 gives the results achieved by clustering the t-SNE embedding of the reduced m-dimensional data. It is noteworthy that the ARI for clustering the t-SNE embedding of the PCA result is higher than the ARI for simply clustering the PCA result. This strongly suggests that using non-linearities (in this case through the means of t-SNE) for dimensionality reduction yields better results.

Clustering method	Dim size (m)	**DiffVAE**	**Disentangled DiffVAE**	**SimpleAE**	**PCA**
K-means	30	0.706 ± 0.002	**0.784** ± 0.005	0.753 ± 0.012	0.697 ± 0.003
	50	0.797 ± 0.001	**0.818** ± 0.002	0.762 ± 0.002	0.699 ± 0.002
GMM	30	0.670 ± 0.007	**0.737** ± 0.007	0.729 ± 0.013	0.714 ± 0.006
	50	0.689 ± 0.012	**0.798** ± 0.009	0.670 ± 0.009	0.724 ± 0.007

Table 6.3: Mean ARI obtained for clustering the t-SNE embedding of the reduced dimension for two setting of the reduced dimension size m and the standard error in the result.

Nevertheless, it is noticeable that ARI is generally lower when performing clustering after applying t-SNE to the reduced dimension computed by the autoencoder models.

6.4 Classifying low-dimensional representation

An additional method for evaluating the quality of the latent dimension produced by the dimensionality reduction methods involves assessing its usefulness as an input to classifier for the cell states. This way, we can analyse whether the reduced dimension captured the most important characteristics about the input data so that it can be used in classification tasks.

Two common classification methods are used: a support vector machine (SVM) and a neural network. The classification methods are applied to the reduced m-dimensional data, for $m = 50$, obtained for the Monocytes Neutrophils, Thrombocytes and Erythrocytes.

SVM Classification method that maximises a soft margin (regulated by hyperparameter C) between the data points and a hyperplane separating them into classes. Optimising the hyperparameter C represents a bias-variance trade-off; C will be optimised in the range $C \in 2^{[-5,10]}$. To perform non-linear classification, the kernel trick is used with the SVM so that the SVM can learn the separating hyperplane in a high dimensional feature space. The Radial Basis Function kernel is used:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \tag{6.2}$$

where γ is optimised in the range $\gamma \in [-10, 5]$. C and γ are optimised through cross-validation.

Neural network Model can be used for classification by having the neurons in the output layer represent the predictions for the possible classes. The softmax activation function is used for the output layer of the neural network to monotonically transform any real-valued vector $\mathbf{y}'$ into a categorical probability distribution:

$$\text{softmax}\,(\mathbf{y}')_i = \frac{\exp(y'_i)}{\sum_{j=1}^{k} \exp(y'_j)}, \tag{6.3}$$

such that the i-th output of the network indicated the probability of input being in the i-th class. As we want the network to classify Monocytes Neutrophils, Thrombocytes and Erythrocytes, we will have 4 neurons in the output layer. Table 6.4 summarises the hyperparameter setting used to train the neural network.

The loss function minimised during training is the cross-entropy between the true label $\mathbf{y}$ and predicted label $\mathbf{y}'$ computed by the network for input $\mathbf{x}$:

$$\mathcal{L}_{\mathrm{NN}}(\boldsymbol{\theta}, \mathbf{x}, \mathbf{y}) = -\sum_{i=1}^{k} y_j \log(y'_j). \tag{6.4}$$

Hyper-parameter	Value setting
Network layers	FC(256) - ReLU activation, FC(512) - ReLU activation, FC(256) - ReLU activation, FC(4) - Softmax activation
Learning rate	0.0001
Batch size	128
Training epochs	300
Optimiser used	Adam

Table 6.4: Hyperparameter setting for the neural network used to classify the mature blood cells. Fully connected (FC) layers are described by their size, i.e FC(512) indicates a fully connected layer consisting of 512 neurons.

Stratified 10-fold cross-validation was used to evaluate the performance of the classification models. This means that at each iteration, 9 folds of the dataset were used for training and one for evaluation. Stratified cross-validation ensures that the folds maintain the proportion of the four classes.

The dimensionality reduction methods were evaluated by varying the reduced dimension size (m). For each setting of m, classification methods were trained and evaluated using stratified 10-fold cross-validation. The results obtained for classifying the reduced dimension of the gene expression data into the four different mature blood cells are illustrated in Figures 6.3 and 6.4.

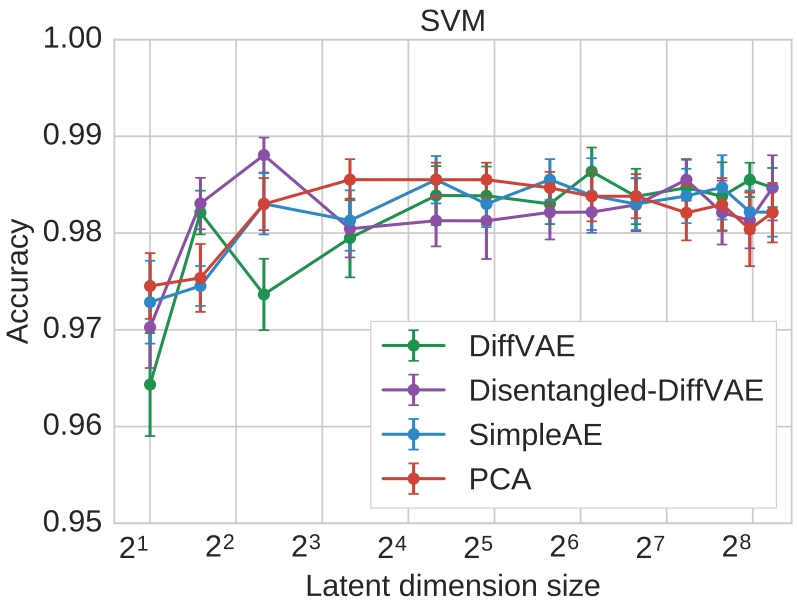


Figure 6.3: Accuracy and the standard error in the results obtained by performing SVM classification with varying the size of the latent dimension.

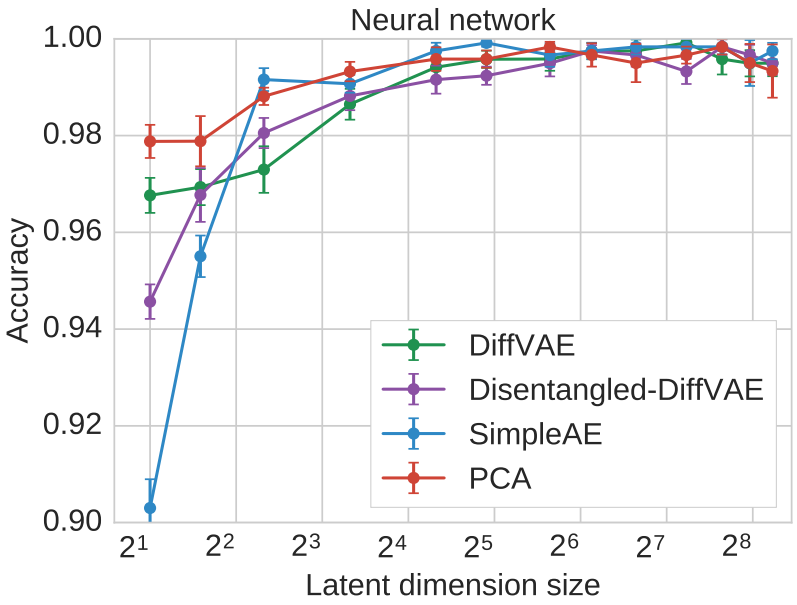


Figure 6.4: Accuracy and the standard error in the results obtained by performing neural network classification with varying the size of the latent dimension.

The results indicate that for a smaller number of dimension, such as $m = 2$, the representation obtained using PCA achieves the best results in the classification task. This may be the case because the first two principle components correspond to the two largest axes of variation in the data. However, as we increase the number of dimensions, all of the dimensionality reduction methods achieve comparable performance on the classification task.

6.5 Summary

This chapter performed a comparative evaluation of DiffVAE, Disentangled-DiffVAE, a simple autoencoder and PCA in terms of their abilities to perform dimensionality reduction. The low-dimensional data representation obtained through Disentangled-DiffVAE resulted in manifold embeddings where the different cell types were separated the best, and it also achieved good performance when applying clustering algorithms to it.

Chapter 7

Conclusions

7.1 Accomplishments

The **project achieved its aim** of building data representations that can help us gain more insight into the process of cell differentiation. The unsupervised neural methods proposed are leveraging the probabilistic framework of the variational autoencoder. Therefore, they are capable of modelling the stochastic behaviour of gene expression across the differentiation of cells and build relevant data representations.

The **methodology developed was successfully applied** on a biological dataset consisting of gene expression information about haematopoietic stem and progenitor cells (HSPCs) and four types of mature blood cells. The results obtained prove the capabilities of the models in answering important biomedical questions. The **methods proposed are general** and can be applied to a wide range of datasets containing gene expression measurements for cells at different stages of in the differentiation process.

This projects makes the following contributions in terms of the unsupervised neural models developed:

- proposes a variational autoencoder architecture suitable for building good representations about cell differentiation from gene expression data: **DiffVAE model**;
- shows how disentanglement methods based on information theory can be used to improve the latent representation built by the variational autoencoder from

gene expression data (first work to apply disentanglement methods for achieving better separation of the biological factors of variation in the gene expression data.): **Disentangled-DiffVAE model**;

- proposes an architecture for a graph variational autoencoder suitable for predicting links between the cells in the dataset (first work to use graph convolutional networks on gene expression data): **Graph-DiffVAE model**;

- performs a comparative evaluation of DiffVAE and Disentangled-DiffVAE with dimensionality reduction techniques such as a simple autoencoder and PCA;

The contributions of the project in terms of the methodology developed for using DiffVAE, Disentangled-DiffVAE and Graph-DiffVAE, as well as important results obtained on the analysed biological dataset are as follows:

- method for identifying latent dimensions in the variational autoencoders modelling biological processes influencing cell differentiation:

  * latent dimensions in DiffVAE and Disentangled-DiffVAE encode the differentiation of the mature blood cells in the dataset.

- method for identifying genes influencing the biological process encoded in the latent representation:

  * DiffVAE and Disentangled-DiffVAE can find driver genes for the differentiation of HSPCs into the mature blood cells.

- method for modifying the latent representation computed for less specialised cell in order to changes them into more specialised cell. The changes in cell typed were labelled by a classifier.

  * showed how to change HSPCs into Neutrophils (a type of mature blood cell) through operations on the latent dimensions encoding the differentiation of Neutrophils.

- method for predicting links between cells based on gene expression data and an initial state of the regulatory dynamics between cells:

  * the links predicted by Graph-DiffVAE between the less specialised cells and the mature blood cells could be interpreted as potential differentiation trajectories for the less specialised cells.

7.2 Future work

The ideas developed in this project suggest many directions for interesting research opportunities that can explored in future work, which involve:

- using **transfer learning methods** to leverage the underlying explanatory factors learnt about the differentiation of some particular cellular types in order to gain knowledge about the differentiation of other types of cells.
- incorporating biological knowledge into the initial graph built from the cells that is given as input to Graph-DiffVAE. This will encourage the prediction of relationships between cells having even more biological relevance.
- building methods in order to understand the latent factors that influence a stem cell to differentiate into a cancerous cell rather than a healthy blood cell, as illustrated in Figure 7.1. In doing so, we can gain better understanding about the the development of cancer caused by changes in gene expression and devise target treatment against it.

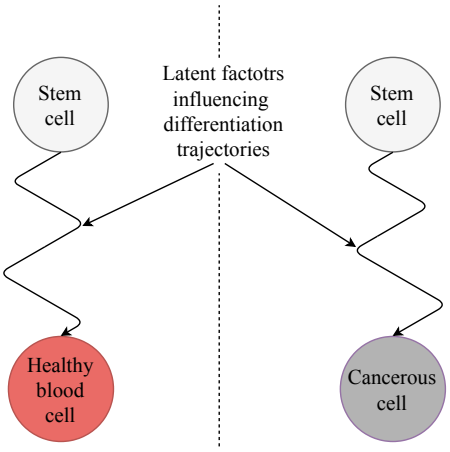


Figure 7.1: Latent factors can affect the differentiation trajectory of a blood stem cell and make it differentiate into a cancerous cell rather a healthy blood cell.

7.3 Final remarks

This project has represented an exciting opportunity to explore unsupervised machine learning techniques in a multidisciplinary setting. A challenging aspect was the need to gain knowledge about biological concepts related to cell differentiation in order to be able to devise appropriate machine learning techniques. However, it is important to point out the benefits of such multidisciplinary work in advancing the field of computer science. In particular, by posing interesting and challenging problems, biomedicine can lead to the development of computational techniques and theoretical knowledge.

The work done in this project resulted in a computational framework capable of building, as well as analysing, data representations, thus having the potential to help research on the biological processes driving cell differentiation that are not yet fully understood.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[5] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[7] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[9] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11,

no. Dec, pp. 3371–3408, 2010.

[10] I. Chambers, J. Silva, D. Colby, J. Nichols, B. Nijmeijer, M. Robertson, J. Vrana, K. Jones, L. Grotewold, and A. Smith, "Nanog safeguards pluripotency and mediates germline development," *Nature*, vol. 450, no. 7173, p. 1230, 2007.

[11] Y. Toyooka, D. Shimosato, K. Murakami, K. Takahashi, and H. Niwa, "Identification and characterization of subpopulations in undifferentiated es cell culture," *Development*, vol. 135, no. 5, pp. 909–918, 2008.

[12] Wikipedia contributors, "Single cell sequencing," 2004. [Online; accessed 02-June-2018].

[13] C. Trapnell, D. Cacchiarelli, J. Grimsby, P. Pokharel, S. Li, M. Morse, N. J. Lennon, K. J. Livak, T. S. Mikkelsen, and J. L. Rinn, "The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells," *Nature biotechnology*, vol. 32, no. 4, p. 381, 2014.

[14] X. Qiu, A. Hill, J. Packer, D. Lin, Y.-A. Ma, and C. Trapnell, "Single-cell mrna quantification and differential analysis with census," *Nature methods*, vol. 14, no. 3, p. 309, 2017.

[15] B. Treutlein, D. G. Brownfield, A. R. Wu, N. F. Neff, G. L. Mantalas, F. H. Espinoza, T. J. Desai, M. A. Krasnow, and S. R. Quake, "Reconstructing lineage hierarchies of the distal lung epithelium using single-cell rna-seq," *Nature*, vol. 509, no. 7500, p. 371, 2014.

[16] R. Durruthy-Durruthy, A. Gottlieb, B. H. Hartman, J. Waldhaus, R. D. Laske, R. Altman, and S. Heller, "Reconstruction of the mouse otocyst and early neuroblast lineage at single-cell resolution," *Cell*, vol. 157, no. 4, pp. 964–978, 2014.

[17] L. Velten, S. F. Haas, S. Raffel, S. Blaszkiewicz, S. Islam, B. P. Hennig, C. Hirche, C. Lutz, E. C. Buss, D. Nowak, *et al.*, "Human haematopoietic stem cell lineage commitment is a continuous process," *Nature cell biology*, vol. 19, no. 4, p. 271, 2017.

[18] Terese Winslow, "Nci dictionary of cancer terms," 2007. [Online; accessed 02-June-2018].

[19] P. Smolensky, "Information processing in dynamical systems: Foundations of

harmony theory," tech. rep., Colorado University at Boulder Department of Computer Scienc, 1986.

[20] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[21] C. Poultney, S. Chopra, Y. L. Cun, *et al.*, "Efficient learning of sparse representations with an energy-based model," in *Advances in neural information processing systems*, pp. 1137–1144, 2007.

[22] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, pp. 153–160, 2007.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[24] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[25] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[26] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," *arXiv preprint arXiv:1505.00687*, 2015.

[27] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1422–1430, 2015.

[28] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *European Conference on Computer Vision*, pp. 835–851, Springer, 2016.

[29] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," in *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.

[30] B. Lee, J. Baek, S. Park, and S. Yoon, "deeptarget: end-to-end learning framework for microrna target prediction using deep recurrent neural networks," in

Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 434–442, ACM, 2016.

[31] M. Kallenberg, K. Petersen, M. Nielsen, A. Y. Ng, P. Diao, C. Igel, C. M. Vachon, K. Holland, R. R. Winkel, N. Karssemeijer, *et al.*, "Unsupervised deep learning applied to breast density segmentation and mammographic risk scoring," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1322–1331, 2016.

[32] G. P. Way and C. S. Greene, "Extracting a biologically relevant latent space from cancer transcriptomes with variational autoencoders," *bioRxiv*, p. 174474, 2017.

[33] J. Tan, J. H. Hammond, D. A. Hogan, and C. S. Greene, "Adage-based integration of publicly available pseudomonas aeruginosa gene expression data with denoising autoencoders illuminates microbe-host interactions," *MSystems*, vol. 1, no. 1, pp. e00025–15, 2016.

[34] S. Rashid, S. Shah, Z. Bar-Joseph, and R. Pandya, "Project dhaka: Variational autoencoder for unmasking tumor heterogeneity from single cell genomic data," *bioRxiv*, p. 183863, 2018.

[35] L. Deng, C. Fan, and Z. Zeng, "A sparse autoencoder-based deep neural network for protein solvent accessibility and contact number prediction," *BMC bioinformatics*, vol. 18, no. 16, p. 569, 2017.

[36] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, pp. 1025–1035, 2017.

[37] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[38] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," *arXiv preprint arXiv:1703.06103*, 2017.

[39] R. van den Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *stat*, vol. 1050, p. 7, 2017.

[40] E. Wahlberg, T. Karlberg, E. Kouznetsova, N. Markova, A. Macchiarulo, A.-G. Thorsell, E. Pol, Å. Frostell, T. Ekblad, D. Öncü, *et al.*, "Family-wide

chemical profiling and structural analysis of parp and tankyrase inhibitors," *Nature biotechnology*, vol. 30, no. 3, p. 283, 2012.

[41] K. Y. Yeung and W. L. Ruzzo, "Principal component analysis for clustering gene expression data," *Bioinformatics*, vol. 17, no. 9, pp. 763–774, 2001.

[42] C. Guibentif, R. E. Rönn, C. Böiers, S. Lang, S. Saxena, S. Soneji, T. Enver, G. Karlsson, and N.-B. Woods, "Single-cell analysis identifies distinct stages of human endothelial-to-hematopoietic transition," *Cell reports*, vol. 19, no. 1, pp. 10–19, 2017.

[43] S. McKinney-Freeman, P. Cahan, H. Li, S. A. Lacadie, H.-T. Huang, M. Curran, S. Loewer, O. Naveiras, K. L. Kathrein, M. Konantz, *et al.*, "The transcriptional landscape of hematopoietic stem cell ontogeny," *Cell stem cell*, vol. 11, no. 5, pp. 701–714, 2012.

[44] Y. Kluger, D. P. Tuck, J. T. Chang, Y. Nakayama, R. Poddar, N. Kohya, Z. Lian, A. B. Nasr, H. R. Halaban, D. S. Krause, *et al.*, "Lineage specificity of gene expression patterns," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 17, pp. 6508–6513, 2004.

[45] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch, "Kernel pca and de-noising in feature spaces," in *Advances in neural information processing systems*, pp. 536–542, 1999.

[46] Z. Liu, D. Chen, and H. Bensmail, "Gene expression data classification with kernel principal component analysis," *BioMed Research International*, vol. 2005, no. 2, pp. 155–159, 2005.

[47] Q. Wang, "Kernel principal component analysis and its applications in face recognition and active shape models," *arXiv preprint arXiv:1207.3538*, 2012.

[48] J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. M. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J. M. Stuart, C. G. A. R. Network, *et al.*, "The cancer genome atlas pan-cancer analysis project," *Nature genetics*, vol. 45, no. 10, p. 1113, 2013.

[49] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[50] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedfor-

ward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[51] F. Li, A. Karpathy, J. Johnson, S. Yeung, and A. TAs, "Cs231n: Convolutional neural networks for visual recognition. 2018," *URL http://cs231n. stanford. edu*.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[54] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[55] A. Makhzani and B. J. Frey, "Pixelgan autoencoders," in *Advances in Neural Information Processing Systems*, pp. 1972–1982, 2017.

[56] H. Kim and A. Mnih, "Disentangling by factorising," *arXiv preprint arXiv:1802.05983*, 2018.

[57] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *Information Theory Workshop (ITW), 2015 IEEE*, pp. 1–5, IEEE, 2015.

[58] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," 2016.

[59] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in beta-vae," *arXiv preprint arXiv:1804.03599*, 2018.

[60] S. Zhao, J. Song, and S. Ermon, "Infovae: Information maximizing variational autoencoders," *arXiv preprint arXiv:1706.02262*, 2017.

[61] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample-problem," in *Advances in neural information processing systems*, pp. 513–520, 2007.

[62] Y. Li, K. Swersky, and R. Zemel, "Generative moment matching networks," in

International Conference on Machine Learning, pp. 1718–1727, 2015.

[63] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani, "Training generative neural networks via maximum mean discrepancy optimization," *arXiv preprint arXiv:1505.03906*, 2015.

[64] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," *arXiv preprint arXiv:1803.10459*, 2018.

[65] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[66] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[67] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online; accessed ¡today¿].

[68] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[69] E. I. Athanasiadis, J. G. Botthof, H. Andres, L. Ferreira, P. Lio, and A. Cvejic, "Single-cell rna-sequencing uncovers transcriptional states and fate decisions in haematopoiesis," *Nature communications*, vol. 8, no. 1, p. 2045, 2017.

[70] M. T. N. Tran, M. Hamada, H. Jeon, R. Shiraishi, K. Asano, M. Hattori, M. Nakamura, Y. Imamura, Y. Tsunakawa, R. Fujii, *et al.*, "Mafb is a critical regulator of complement component c1q," *Nature communications*, vol. 8, no. 1, p. 1700, 2017.

[71] L. M. Kelly, U. Englmeier, I. Lafon, M. H. Sieweke, and T. Graf, "Mafb is an inducer of monocytic differentiation," *The EMBO journal*, vol. 19, no. 9, pp. 1987–1997, 2000.

[72] P. Patil, T. Uechi, and N. Kenmochi, "Incomplete splicing of neutrophil-specific genes affects neutrophil development in a zebrafish model of poikiloderma with neutropenia," *RNA biology*, vol. 12, no. 4, pp. 426–434, 2015.

[73] M. J. Foulkes, K. M. Henry, J. Rougeot, E. Hooper-Greenhill, C. A. Loynes, P. Jeffrey, A. Fleming, C. O. Savage, A. H. Meijer, S. Jones, *et al.*, "Expression and regulation of drug transporters in vertebrate neutrophils," *Scientific reports*, vol. 7, no. 1, p. 4967, 2017.

[74] E. A. Harvie and A. Huttenlocher, "Neutrophils in host defense: new insights from zebrafish," *Journal of leukocyte biology*, vol. 98, no. 4, pp. 523–537, 2015.

[75] W. Pimtong, M. Datta, A. M. Ulrich, and J. Rhodes, "Drl. 3 governs primitive hematopoiesis in zebrafish," *Scientific reports*, vol. 4, p. 5791, 2014.

[76] F. E. Moore, E. G. Garcia, R. Lobbardi, E. Jain, Q. Tang, J. C. Moore, M. Cortes, A. Molodtsov, M. Kasheta, C. C. Luo, *et al.*, "Single-cell transcriptional analysis of normal, aberrant, and malignant hematopoiesis in zebrafish," *Journal of Experimental Medicine*, pp. jem–20152013, 2016.

[77] G. Khandekar, S. Kim, and P. Jagadeeswaran, "Zebrafish thrombocytes: functions and origins," *Advances in hematology*, vol. 2012, 2012.

[78] N. Bushati, J. Smith, J. Briscoe, and C. Watkins, "An intuitive graphical visualization technique for the interrogation of transcriptome data," *Nucleic acids research*, vol. 39, no. 17, pp. 7380–7389, 2011.

[79] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[80] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, pp. 849–856, 2002.

[81] E. Taskesen and M. J. Reinders, "2d representation of transcriptomes by t-sne exposes relatedness between human tissues," *PloS one*, vol. 11, no. 2, p. e0149853, 2016.

[82] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

Appendix A

Variational autoencoder objective - derivations

A.1 Evidence lower bound

The evidence lower bound (ELBO) maximised by the variational autoencoder is given by:

$$\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})). \tag{A.1}$$

The log likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ can be rewritten as follows:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \left(\int_{\mathbf{z} \in \mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \mathrm{d}\mathbf{z} \right) \log p_{\boldsymbol{\theta}}(\mathbf{x}) \\
&= \int_{\mathbf{z} \in \mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}) \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z} \in \mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})} \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z} \in \mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})} \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z} \in \mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} + \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})} \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z} \in \mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})).
\end{aligned} \tag{A.2}$$

Substituting the expression obtained for $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ in Equation A.1 we obtain:

$$\begin{aligned}
\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \int_{\mathbf{z}\in\mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z}\in\mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z}\in\mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z}\in\mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \mathrm{d}\mathbf{z} - \int_{\mathbf{z}\in\mathcal{Z}} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left(\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})) \qquad \text{(A.3)}
\end{aligned}$$

A.2 Regularisation term

Let the distribution of the latent variables $q_{\boldsymbol{\phi}}(\mathbf{z}) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})]$ and let $q(\mathbf{x}, \mathbf{z}) = p_{\text{data}}(\mathbf{x}) q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ be the joint distribution of the data and of the encoding distribution.

When training the variational encoder, the regularisation term $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}))$ is computed over expectation over the training data, such that:

$$\begin{aligned}
\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}))] &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})} \left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{x})} \frac{q_{\boldsymbol{\phi}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})} \left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{x})}\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})} \left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})} \left[\log \frac{q(\mathbf{z}, \mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{x}) p_{\text{data}}(\mathbf{x})}\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{z})} \left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right] \\
&= I(\mathbf{x}; \mathbf{z}) + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}) || p_{\boldsymbol{\theta}}(\mathbf{z})) \\
&\geq I(\mathbf{x}; \mathbf{z}), \qquad \text{(A.4)}
\end{aligned}$$

where $I(\mathbf{x}; \mathbf{z})$ is the mutual information defined under joint distribution of data and encoding distribution.