STATEMENT

Name: Aurél György Prósz

Neptun code: XGRP0J

ELTE Faculty of Science, Field of studies: Department of Physics of Complex Systems, Physics

Title of thesis: Predicting single-cell perturbation responses with Generative Adversarial Neural Networks

As the author of this thesis I declare that it is my own work and that I have acknowledged and referenced the work and ideas of others.

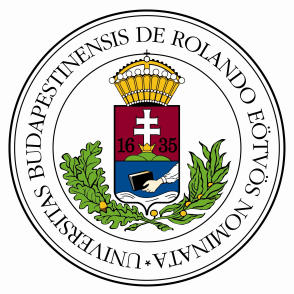
01.02.2020 **....., Budapest**

Prósz György Aurél

signature

Predicting single-cell perturbation responses with Generative Adversarial Neural Networks

Master Thesis



Aurél György Prósz

Department of Physics of Complex Systems

Eötvös Loránd University, Budapest

Supervisor

Prof. István Csabai

In partial fulfillment of the requirements for the degree of

Master of Science in Physics

January 02, 2020

Acknowledgements

I would like to express my gratitude to my supervisor Dr. István Csabai for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore I would like to thank Dezső Ribli, Bálint Pataki and András Biricz for introducing me to the topic as well for the support on the way. I am also grateful to Dr. Kristóf Szalay and Sebestyén Kamp for spending their time reading this thesis and providing useful suggestions about it.

Abstract

Generative Adversarial Networks (GANs) are deep neural net architectures comprised of two nets, pitting one against the other in a game-theoretical sense. Notably, only the discriminator directly observes real data while the generator improves its simulations through interaction with the discriminator. As training progresses both neural networks learn key features of the training data.

Recent advances have enabled gene expression profiling of single cells at a lower cost. As more data is produced there is an increasing need to integrate diverse data sets and better analyse underutilised data to gain biological insights.

In this thesis I propose a novel GAN architecture named scsGAN (Single Cell State GAN), which is shown in this work to be able to approximate single-cell RNAseq data distribution better than conventional methods, and can be used to perform various complex tasks, such as data augmentation, realistic interpolation between data points and latent space vector arithmetics to predict single-cell perturbation responses.

There is an upcoming availability of large-scale single-cell atlases of organs in healthy and perturbed states, and the results suggest scsGAN could become a tool for experimental design in the context of disease and drug treatment.

Contents

Chapter 1

Introduction

1.1 Motivations

Artificial intelligence, and more specifically Deep Learning (DL) is playing an increasingly important role in our everyday lives. It has already made a huge impact in a range of areas, such as cancer diagnosis, precision medicine, self-driving cars, predictive forecasting, speech recognition, and speech synthesis and experts say it is perhaps the most significant development in the field of computer science in recent times. It is already disrupting and transforming businesses and industries and there is an accelerating race among the world's leading economies and technology companies to advance deep learning. In the far future one of the goals of this race is to create the holy grail of the artificial intelligence field, the Artificial General Intelligence (AGI), which is a machine that can understand or learn any intellectual task that a human being can.

The deep learning technique is inspired by the human nervous system and the architecture of the brain and in simplistic terms it consists an input, a multiple hidden, and an output layer consisting of individual processing units called neurons. The main advantages of this method are to accurately extract higher

representation from the dataset which it was trained on and with the help of the so-called backpropagation algorithm it is relatively straightforward to train compared to other machine learning methods. Neural Networks can be used in a variety of problems including pattern recognition, classification, clustering, dimensionality reduction, computer vision, natural language processing (NLP), regression, predictive analysis, data augmentation, and sample generation, and so on [1], [2].

Cross fertilization between the physics and machine learning disciplines dates back decades. Especially statistical physicists made important contributions to the theoretical understanding of statistical learning. The connection between learning theory and statistical mechanics started when statistical learning from samples took over the logic and rule based AI, in the mid-1980s.

Two seminal papers marked this transformation, one of them was Hopfield's neural network model of associative memory which sparked the rich application of concepts from spin glass theory to neural network models. A much tighter application to learning models was made by the seminal work of Elizabeth Gardner who applied the replica trick to calculate volumes in the weights space for simple feed-forward neural networks, for both supervised and unsupervised learning models.

The statistical physics of learning reached its peak in the early 1990s, but the majority of the machine learning field was focused on general input-distribution-independent generalization bounds, characterized by e.g. the Vapnik-Chervonenkis dimension or the Rademacher complexity of hypothesis classes [3].

A prominent example of the above cross fertilization between physics and machine learning is the so-called Restricted Boltzmann Machines (RBMs), which are neural networks that belong to so-called Energy Based Models and they try to minimize a predefined energy function. They use partition functions seen in

statistical physics [4].

Experiments from all areas of physical sciences generate huge amounts of new data, which are also needed to be analyzed. Similar analysis which is found in this thesis can be also applied to data generated from astronomical surveys [5],[6] and particle physics [7].

Even though all cells in our body share nearly identical genotypes, transcriptome information in any one cell reflects the activity of only a subset of genes. This phenomenon states a challenge in the mapping of the genotypes to phenotypes in biology and medicine, and a powerful strategy for tackling this problem is performing transcriptome analysis on single cell level. The many diverse cell types in our body each express a unique transcriptome and conventional bulk population sequencing can provide only the average expression signal for an ensemble of cells and evidence further suggest that gene expression is heterogeneous, even in similar cell types. The sequencing an entire transcriptome at the level of a single-cell was first demonstrated in 1992, and since then with the rise of the next-generation sequencing methods, there has been an explosion of interest in obtaining high-resolution views of single-cell heterogeneity on a global scale. The ability to find and characterize outlier cells within a population has potential implications for furthering our understanding of drug resistance and relapse in cancer treatment [8], [9].

Various deep learning architectures were proposed in the last ten years to advance progress in the field of single cell transcriptomics. RNA sequencing (RNAseq) uses the capabilities of high-throughput DNA sequencing methods to provide insight into the transcriptional state of a biological sample. Traditional methods of high dimensional analysis like linear regression or Random Forest are challenging to use on RNAseq datasets because of the high feature to sample ratio and the high noise levels in biological systems. Modern machine learning and deep learning

approaches have also begun to surmount these obstacles in RNAseq datasets, and novel, powerful generative models such as variational autoencoders (VAEs) and generative adversarial networks (GANs) are used to model the data distribution of the biological samples [10], [11].

1.2 Objectives and Contributions

The main objectives of this thesis are listed below:

- Create a comprehensive review of the state-of-the-art methods of the single-cell RNA sequencing experimental and simulation field.
- Validate that the proposed method using Generative Adversarial Networks can accurately model scRNA-seq data.
- Create a workflow with the GAN model, which can be used to accurately predict arbitrary single-cell transcriptomics perturbations, possibly leading to more cost-effective in-vitro experiments in single-cell biology and drug discovery.

1.3 Overview of the Thesis

The thesis is separated into four main parts:

- **Theoretical background:** In this section, I introduce the theoretical background of the artificial neural networks and the techniques of how these kinds of machine learning algorithms can be trained on the data. Next, I discuss various generative models used to learn or approximate the real data distribution focused heavily on GANs. In the second part of this section I explore how experimental techniques are used to quantify multiple

gene expression in individual cells. Last, but not least I introduce the most common tools and methods to analyze single cell RNAseq data.

- **The scsGAN model:** I introduce the proposed GAN model and its architecture with the relevant information on how to train and evaluate its results.
- **Methods:** In this section, I discuss how the in-silico experiments were performed and evaluated.
- **Results:** Discussion of the results of the in-silico experiments, and the planning of the future extensions or validation in-vitro experiments.

My own contribution to this topic is introduced after the description of the scsGAN model in Chapter 3.

Chapter 2

Theoretical background

This chapter provides the theoretical background on various concepts included in this thesis. First, I begin describing artificial neural networks, the primary machine learning technique used through the analysis. Next, I introduce generative models that can learn any kind of data distribution using unsupervised learning and then generate new samples from this distribution with some variation.
Later in this chapter I describe the theoretical background of the so called generative adversarial networks (GANs), which is one of the most efficient and commonly used techniques on generative tasks. There are many types of GANs, I compare the most frequently used versions and compare their advantages and disadvantages.
In the last part, I introduce how multiple gene expression could be quantified in individual cells. Both in-vitro and data processing techniques are characterized.

2.1 Artificial Neural Networks

The first abstract mathematical models on how the brain processes information were proposed in the late 50's. According to the models, the neurons and synapses

in the brain give rise to the so called neural networks, by connecting the neurons to each other with synapses, allowing information to flow. These attempts to try to describe the information processing mechanism of the brain paved the way to the field of Artificial Neural Networks (ANN) [12].
First, I describe how a single artificial neuron processes information. An artificial neuron can be characterized by three parameters [13]:

- The weight vector, which is an n- dimensional vector of real numbers, $\boldsymbol{w} \in \mathbb{R}^n$.
- The bias parameter, $b \in \mathbb{R}$.
- An activation function $\Phi : \mathbb{R} \to \mathbb{R}$.

  The output of the artificial neuron is defined as:

$$y = \phi\left(\mathbf{w} \cdot \mathbf{x} + b\right)$$

  where $\mathbf{x} \in \mathbb{R}^n$ serving as the input vector of the neuron.

  The most widely used activation functions are the *linear*, *sigmoid*, *tanh* and the *leaky RELU*. The Table 2.1 summarizes these activation functions.

Table 2.1. Definitions of the most widely used activation functions.

Activation function	Equation
Linear	$\phi(x) = x$
Sigmoid	$\phi(x) = \frac{1}{1+e^{-x}}$
Tanh	$\phi(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
Leaky RELU	$\phi(x) = max(\alpha x, x)$

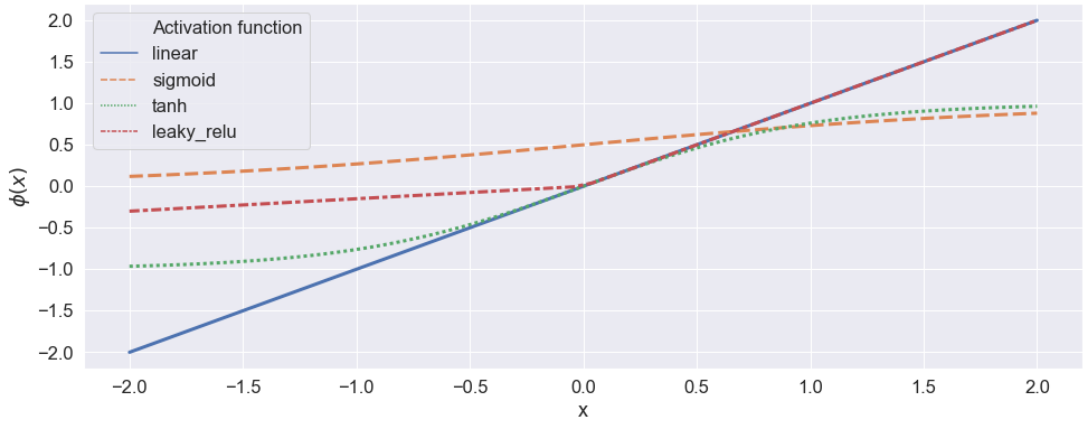


Figure 2.1. The activation functions compared with each other. The leaky RELU function was calculated with $\alpha = 0.15$.

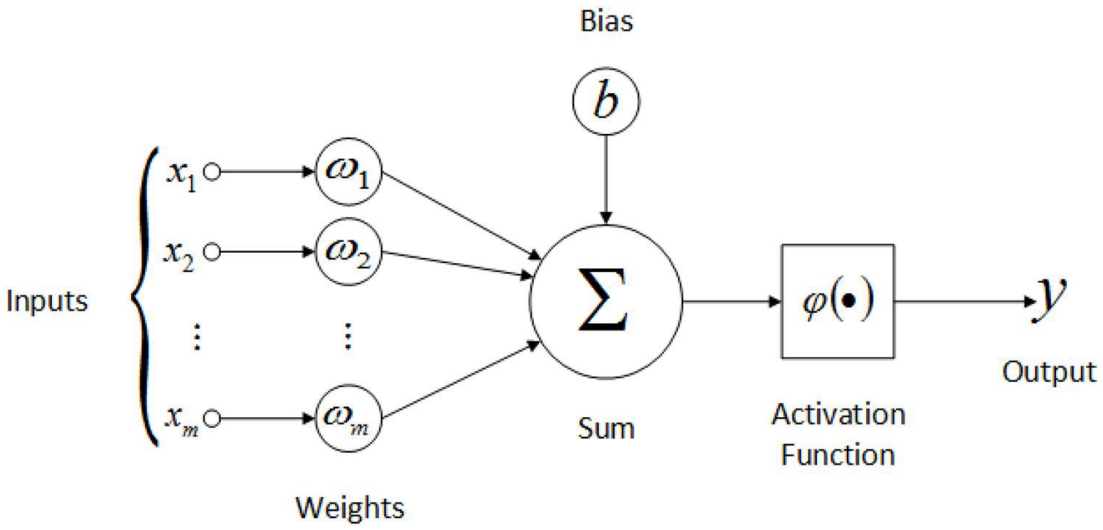


Figure 2.2. Schematic representation of a single artificial neuron. The image was taken from [14].

By taking these single neurons, one can construct an ANN by connecting them together using the outputs of the previous neurons to be the inputs of the next one. If such a network exhibit loops, then it is considered a recurrent neural network (RNN), and feedforward neural network if not. A feedforward ANN can consist of several layers of neurons. The first layer of nodes in this network is

fed with the input data, and is called the *input layer*. The output of this layer is then fed to the first layer of the *hidden layers*. The output of the first hidden layer can serve as the input of the next hidden layer, and so on.
By stacking more hidden layers after each other, the network can be mode **deeper**, and by increasing the number of neurons in a given layer, the network is getting **wider**. In a *fully connected ANN*, every neuron is connected to every other in the next layer. The last layer, which output is not connected to any other layer is called the *output layer*. The output of the n-th hidden layer can be calculated as:

$$\mathbf{y}^{(n)} = \phi^{(n)} \left(\mathbf{W}^{(n)T} \cdot \mathbf{x}^{(n-1)} + \mathbf{b}^{(n)} \right)$$

where $\phi(\mathbf{x})$ is applied element-wise. By constructing an ANN in this way it has been shown that a multi-layer perceptron with only one hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function. Consequently, this property of the multilayer perceptron is useful in the context of statistical machine learning, because there are well-defined ways on how to adjust the weights and the bias terms in the network to approximate a large class function. The next section gives an overview of these methods [15].

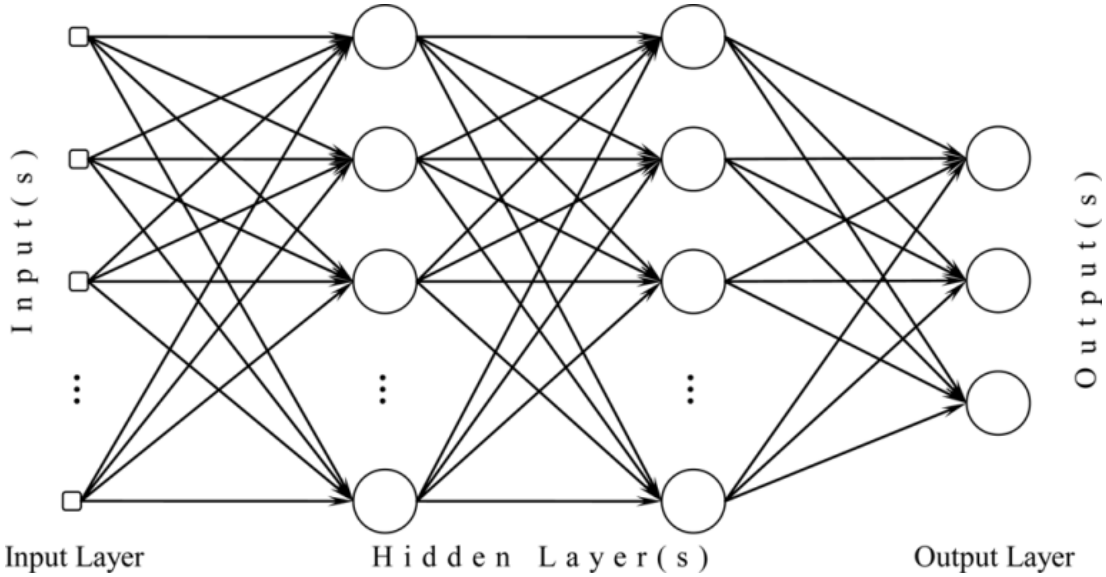


Figure 2.3. Schematic representation of a multilayer perceptron [16].

2.1.1 Learning in neural networks

One of the processes to adjust the weights and bias terms in the network is called *supervised learning* when the ground truth or labels associated to the functional values t_n for a set of input vectors $\mathbf{x}_n$ is known. There are two main steps before the optimization process begins: choosing a suitable activation function and then define a *loss function*, which value needs to be maximized or minimized by changing the ANN parameters. Selecting the appropriate loss function is a critical step in any of the machine learning problems, and below is a short summary on the most basic and widely used loss functions based on what type of task are they the most suitable for. The two main tasks in machine learning are:

- **Regression:** Regression predictive modeling is about approximating a mapping function f from input variables X to a **continuous** output variable $\hat{y}$. A continuous output variable is a real-value, such as an integer or floating

point value. These are often quantities, such as amounts and sizes. One of the simplest loss functions for this task is the Root Mean Squared Error (RMSE). RMSE is a quadratic scoring rule that also measures the average magnitude of the error. $RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y_j})^2}$, where n is the total number of observations, $\hat{y}$ is the predicted values, and y is the true values. Another useful loss function is the Mean Absolute Error (MAE), which measures the average sum of the absolute difference between the actual value and the predicted value for all data points, $MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y_j}|$. Both MAE and RMSE express average model prediction error in units of the variable of interest, and both metrics are indifferent to the direction of errors. The main difference between them is RMSE has the benefit of penalizing large errors more since the errors are squared before they are averaged, this way the RMSE gives relatively high weight to higher errors.

- **Classification:** Classification predictive modeling is the task of approximating a mapping function f from input variables X to **discrete** output variables y. These output variables are often called labels, categories or classes. The mapping function predicts the class or category for a given observation. Log-loss (or binary cross-entropy) is a simple loss function for a binary classification task. It is a measurement of accuracy that incorporates the idea of probabilistic confidence given by the following expression for binary class. Let H the Log-loss value, then:
  $H = -\frac{1}{n}\sum_{j=1}^{n} y_j \cdot log(\hat{y_j}) + (1 - y_j \cdot log(1 - \hat{y_j}))$ For any given problem, a lower log-loss value means better predictions.

After selecting an error function, we then aim to find the set of parameters that minimizes this error function. This concept can be interpreted spatially by imagining a **parameter space** whose dimensions are the values of each of the model parameters, and for which the error function will form a surface of varying height

depending on its value for each parameter. Model training is thus equivalent to finding a point in parameter space that makes the height of the error surface small.

Depending on the problem, the error surface can be very complex with sharp edges, local minimas and saddle points (an example can be seen in Figure 2.4). By incrementally lowering the value of the error function by a method called gradient descent, one can find the desired minima. Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in R_d$ by updating the parameters in the opposite direction of the gradient of the objective function: $-\nabla\eta J(\theta)$ with respect to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley. The basic concept of gradient descend is described in Figure 2.5 [13].

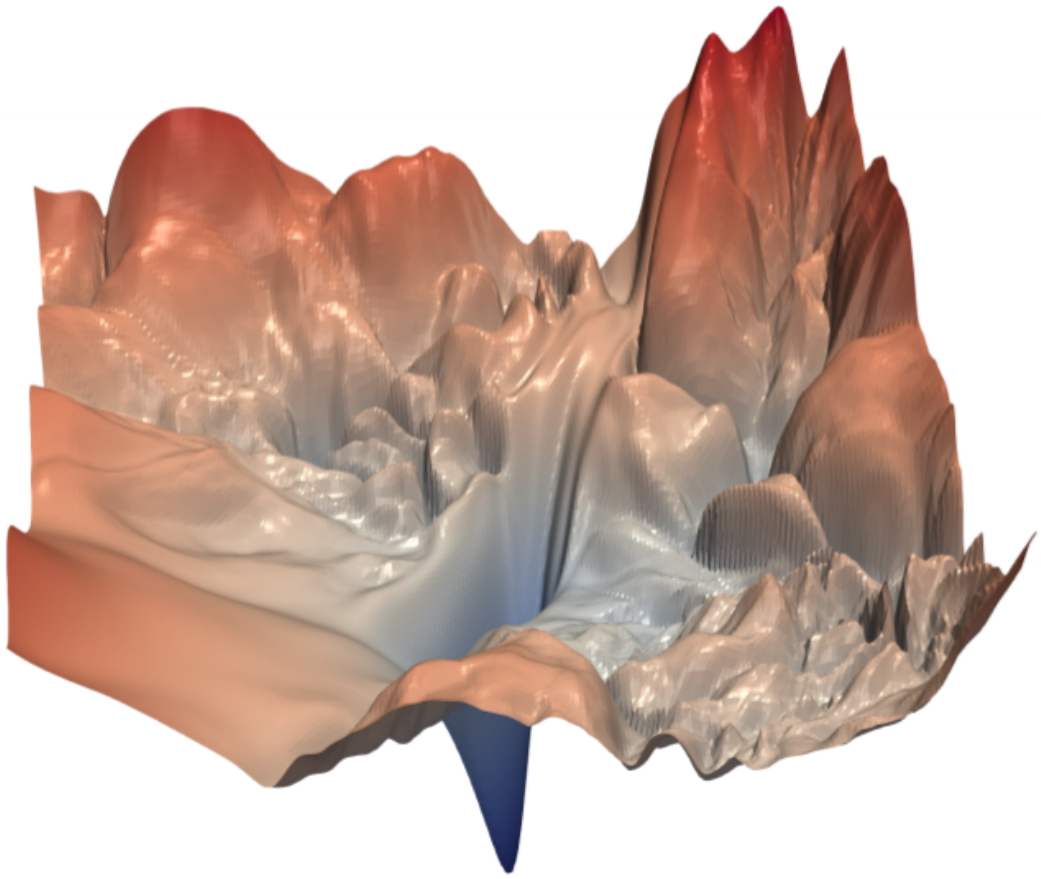


Figure 2.4. An example of a complex error surface. The first two axes represent the first two parameters of the system, the height of the function is the value of the error function [17].

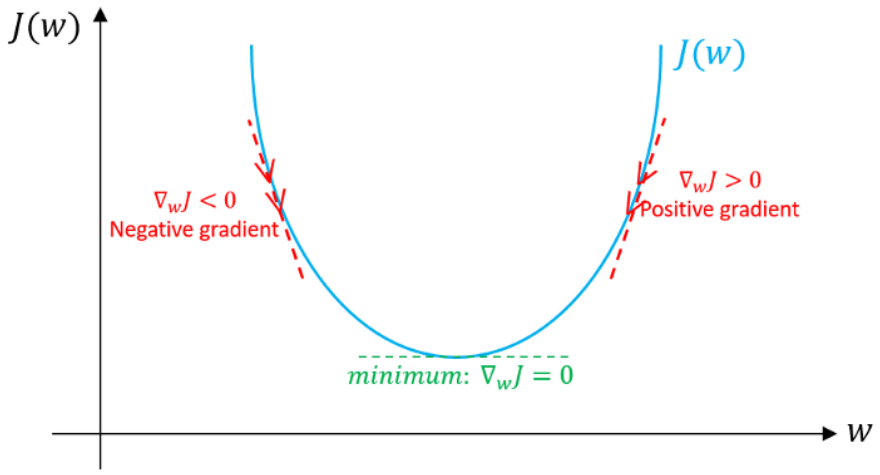


Figure 2.5. Gradient descent works by taking small steps into the direction of the partial derivative of the loss-function, until the minima is reached [18].

2.1.1.1 Backpropagation

To calculate the derivatives in an ANN, a method called **backpropagation** is used. First, the information flows from the input to the output in the ANN model, and then in a reverse direction. The difference of the predicted output and the actual output (the error term), is then determined. The error made by the network to predict the true value or the label is differentiated with respect to all the parameters and weights of the network and then the weights are adjusted accordingly. A more detailed description can be seen in Figure 2.6 [19].

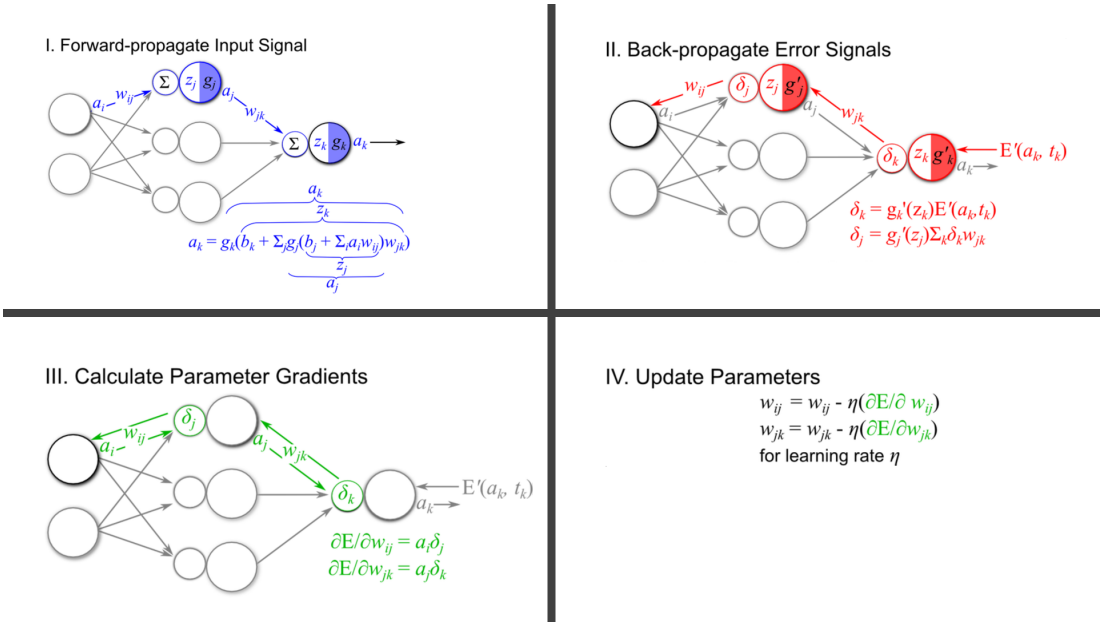


Figure 2.6. The different steps of the backpropagation algorithm detailed. I. A forward propagate signal results in a prediction, and the error is calculated. II. The error signal is backpropagated. III. Using the error terms, the parameter gradients are calculated. IV. Update the parameters for all weights. The figures were taken from [20].

2.1.1.2 Optimizers

Depending on how much data one uses to compute the gradients of the objective function of the ANN, there are three main variants of gradient descent. By selecting one of them, one make a trade-off between the accuracy of the parameter update, and it takes to perform an update.

Batch gradient descent:

Vanilla gradient descent, or batch gradient descent, computes the gradient of the cost function with respect to the parameters for the entire training dataset. As one needs to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory. Batch gradient descent also does not to update the model

online. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

Stochastic gradient descent: Stochastic gradient descent in contrast to batch gradient descent performs a parameter update for each training example. Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. Stochastic gradient descent does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online. While batch gradient descent converges to the minimum of the basin the parameters are placed in, the fluctuation of Stochastic gradient descent, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum. However, it has been shown that when we slowly decrease the learning rate, Stochastic gradient descent shows the same convergence behavior as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

Stochastic gradient descent has trouble navigating ravines, in areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, it oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. *Momentum* is a method that helps accelerate SGD in the relevant direction on the error surface and dampens oscillations. *RMSProp (Root Mean Square Propagation)* also tries to dampen the oscillations, but in a different way than momentum. RMS prop also takes away the need to adjust the learning rate, and does it automatically. More so, RMSProp chooses a different learning rate for each parameter. RMSprop updates the weights with the following algorithm

for each parameter w^j:

$$\nu_t = \rho\nu_{t-1} + (1-\rho)g_t^2$$

$$\Delta\omega_t = -g_t \frac{\eta}{(\nu_t + \epsilon)^{\frac{1}{2}}}$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

where η is the initial learning rate, ν the exponential average of squares of gradients, g_t the gradient at time t along ω^j, ρ is a parameter for the strength of the exponential decay, and ϵ is a smoothing term[21], [22].

2.1.1.3 Weight initialization

Before the training process one has to assign some initial values to the weight vector and the bias terms. This initialization step plays an important role in optimization and the training time of the network. We have to assign some initial values to weight vector W and bias term b before we start our training process. This initialization of weights also plays a vital role in optimization and the training time of the network. The direct advantage of a good initialization is that the network converges quickly. Normally one can assign the random values to the parameters in the beginning. However, assigning extreme values might lead the network to always predicting the same labels the signal. Weight initialization is still an active area of research, with a lot of competing methods [23].

2.2 Generative models

An important class of deep neural networks is the generative models. Generative modelling is an active area of deep learning research that centers around realistic data generation. The goal of a generative model is to learn the density function that describes the original sample data by giving the model the original training

samples, and then use this estimated density to generate fake yet realistic looking data, very similar to the original sample, but which are not exactly the same. Maximum likelihood can be identified as a central task around which generative models can be compared. However, not all generative models use maximum likelihood.

$$\theta^* = argmax\mathbb{E}_{x\sim\rho_{data}} logP_{model}(x|\theta)$$

Here the x describes the input, $P_{model}(x|\theta)$ is a density function, θ is the parameter of this density function. Maximum likelihood consists of measuring the log probability that the density function assigns to all data points and adjusting the parameter θ to increase that probability.

The first large class of the generative models is the explicit density function models. This type of models defines a P_{model}, are able to evaluate it and varies its parameters regarding the training data. However, most of the real world data density functions, such as over speech and images, are intractable and hard to define. In these cases an approximate density function is used, which then divides the class into tractable density and approximate density function models. Examples of the tractable densities are the Fully Visible Belief Nets, and a family of models utilizing change of variables.

The first one make use of an explicit formula based on the chain rule of probability to decompose the probability distribution over a vector into a product over each of the members of the vector:

$$P_{model}(x) = P_{model}(x_1)\prod_{i=2}^{n} P_{model}(x_1|x_i, ..., x_{i-1})$$

In the case of the second one we begin with a simple distribution like a Gaussian and then use a nonlinear function to transform that distribution into another space.

Approximate density models are explicit density models which have intractable density functions but then use tractable approximations to these density functions. The most prominent member of this class is the Variational Autoencoders (VAE). The basic idea of VAE is to write the density function in the following form:

$$logp(x) \geqslant logp(x) - D_{KL}((q(z)||p(z|x)) = \mathbb{E}_{z\sim q} logp(x,z) + H(q)$$

The variable z is a vector of latent variables that provide a hidden code describing the input data. The problem of intractability means variational approximation must be used which introduces a distribution q over the latent variable z to the extent that this distribution q is closer to the true posterior over the latent variable. The distribution q enables to make a bound that becomes smaller and does a better job of establishing a lower bound that defines the true density. Variational encoders therefore define a lower bound. The disadvantage of the VAEs is they tend to be asymptotically inconsistent unless q and thus the lower bound is near perfect [24].

The second class of the generative models do not need an explicitly defined density function to be trained and instead they offer a way to train the model while interacting only indirectly with P_{model}, usually by sampling from it. The most prominent member of this family is the Generative Adversarial Networks, which I will introduce in more detail.

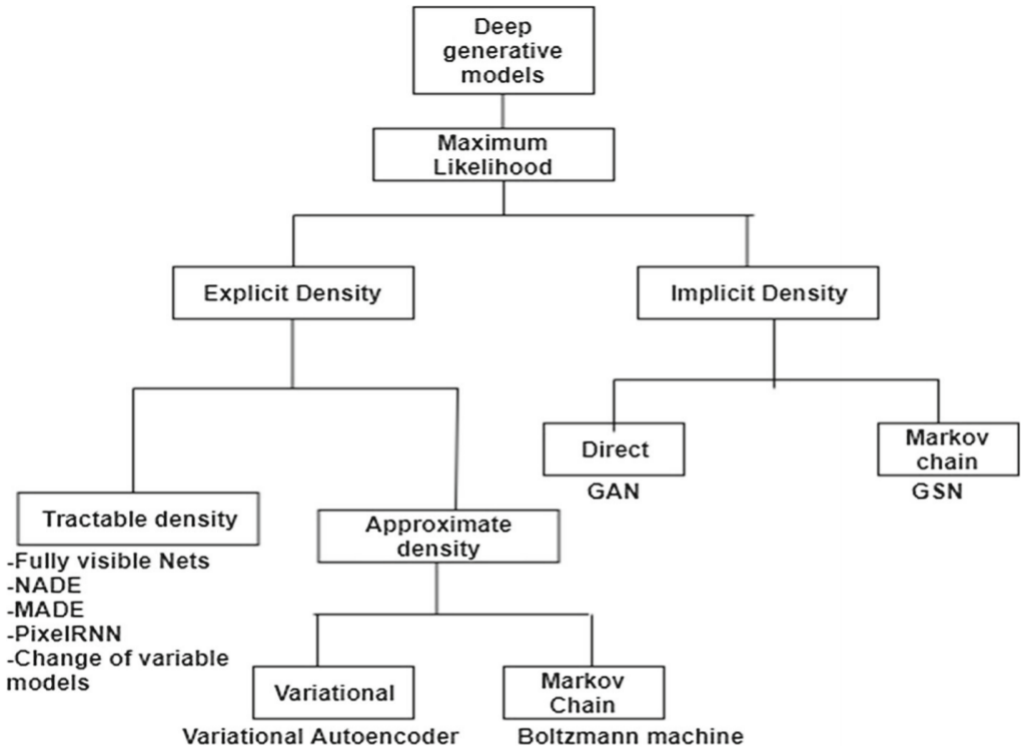


Figure 2.7. Taxonomy of generative models. The image was taken from [24].

2.2.1 Generative Adversarial Networks

The basic architecture of a GAN consists of two neural networks existing as adversaries of each other in the sense of game theory. A well defined loss (or payoff) function defines the operation of the two networks [25].

The first model, which is implemented as a neural network is called the **Generator**. It is the model which actively generates the samples, which are intended to resemble the original samples found in the train set. We can describe the Generator in the following formulation:

$$x = G(z; \theta_G)$$

Here x is the observed variable generated by the Generator, z is the latent space vector encoding the latent code, θ_G are the parameters of the Generator model. Also, it is important to ensure that x has a higher dimension than z. This ensures that the generator is not forced to generate a low dimensional manifold within the space of x.

The second model is called the **Discriminator** and the primary role of it is to inspect a sample to see whether the sample looks real or fake. In the classical GAN, presented by [26], the discriminator network goal is to output a value near 1 for data from the original training set and values close to 0 for data from the Generator.

It should be noted, that both the Generator and the Discriminator model can be any kind of differentiable function that has parameters we can learn using gradient descent.

Training a GAN consists of the following steps: First, sampling a minibatch from both the training set and the generated samples, and then run the Discriminator network on those inputs. The sampling from the Generator network is utilized by sampling the latent vector z from the prior distribution over the latent variables, where z is essentially a vector of unstructured noise, which allows the Generator to output a wide variety of different vectors. Next we apply the Generator network to this latent vector z, which generates a generator sample from it, which can be fed to the Discriminator network.

The discriminator outputs then is a binary classification of real or fake. The error loss on the discriminator's output is the cross entropy cost function. This error is then backpropagated to both the generator and the discriminator networks. Training theoretically stops when the discriminator can no longer discriminate between generated data and training data.

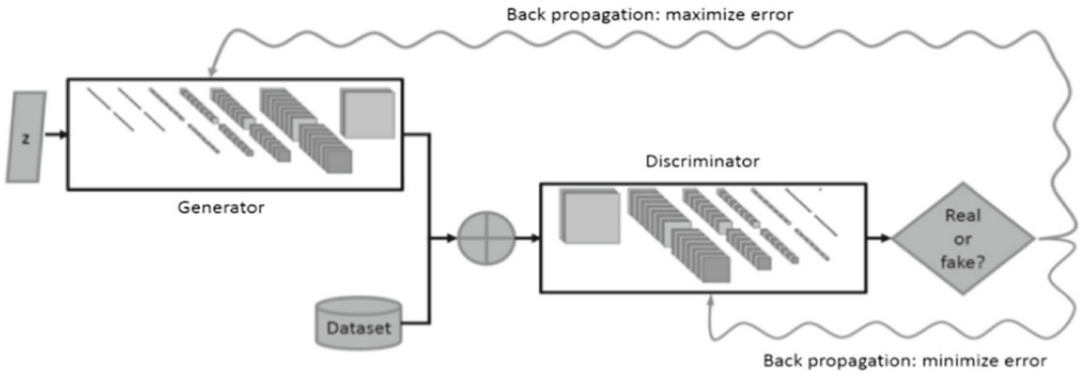


Figure 2.8. Schematic representation of a basic GAN architecture. The image was taken from [27]

.

The loss function of the basic GAN consists of two parts, one for the generator and one for the discriminator:

$$J^D = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} logD(x) - \frac{1}{2}\mathbb{E}_{x\sim p_{data}} log(D(G(x)))$$

$$J^G = -J^D$$

The first equation can be viewed as the cross-entropy between the predictions of the discriminator and the correct labels in the binary classification task discriminating between real and fake data. The second equation simply minimizes the log probability of the discriminator cost function. The minimization of these cost functions can be thought as a minimax game between the two networks. One of the reasons GANs are quite straightforward to train is that we never actually try to infer the probability distribution $p(z|x)$, instead we sample values of z from the prior and then we sample values of x from $p(x|z)$.

Although the results generated by GANs can be remarkable, it can be challenging to train a stable model. The reason is that the training process is inherently un-

stable, resulting in the simultaneous dynamic training of two competing models. This problem has led to introducing new architectures and loss functions, which can stabilize the training of the models. Another major issue is there is no direct correlation between the loss function and the quality of the generated data [24].

2.2.1.1 Wasserstein GAN

The Wasserstein GAN (WGAN) is an extension of the basic concept of GANs, and it seeks an alternative way to training the generator part of the model. The main idea is to replace the method of the Discriminator (which is often called the **Critic** when using Wasserstein GANs) to differentiate between real or fake samples. The basic Discriminator model predicts the probability of a sample being fake or real, but a Critic predicts a value of how fake the sample looks like. It transforms the minimization task to more aligns with the theoretical argument that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples. This way the entire training process is more stable, and the loss of the discriminator appears to relate to the quality of the samples created by the generator.
The main differences between the regular GAN and the WGAN are below:

- The activation function on the output layer of the Discriminator is a linear function.
- Use Wasserstein loss to train the Discriminator and Generator models.
- Updates the Discriminator model more times than the Generator each iteration.

The WGAN method proposes a new cost function, using Wasserstein distance that has generally a smoother gradient everywhere compared to the KL or JS divergence the regular GAN uses.

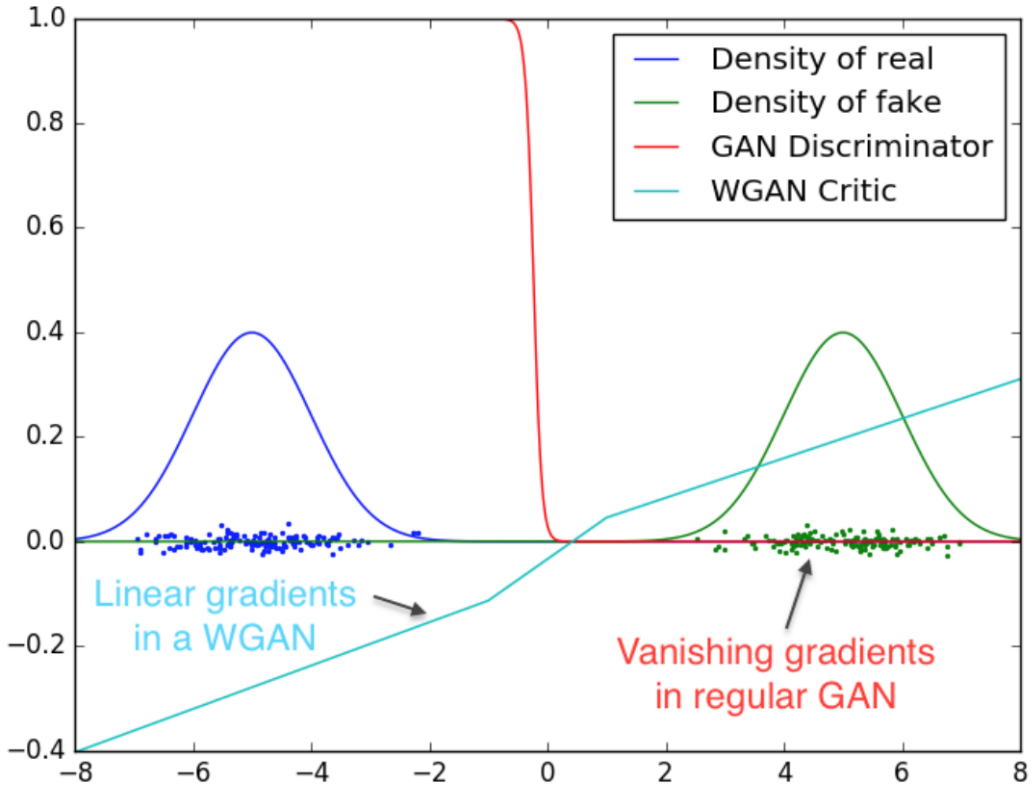


Figure 2.9. The value of the Discriminator's output for both GAN and WGAN. For GAN (the red line), it fills with areas with diminishing or exploding gradients. For WGAN (the blue line), the gradient is smoother everywhere and learns better even the generator is not producing good samples [28].

The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution q to the data distribution p. The Wasserstein distance for the real data distribution P_r and the generated data distribution P_g is mathematically defined as the greatest lower bound (infimum) for any transport plan:

$$W(P_r, P_g) = \inf_{\gamma \prod(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}(||x - y||)$$

This equation for the Wasserstein distance is highly intractable. Using the *Kantorovich-Rubinstein duality*, we can simplify the calculation to:

$$W(P_r, P_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim P_r}(f(x)) - P_\delta(f(x))$$

In the WGAN model the **1-Lipschitz function** is learned by the Discriminator network. To enforce the constraints to learn this function, the Discriminator uses gradient penalty. A differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere, so the WGAN model penalizes if the gradient norm moves away from its target norm value 1:

$$L = \mathbb{E}_{\hat{x}\sim P_g}(D(\hat{x})) - \mathbb{E}_{\hat{x}\sim P_r}(D(x)) + \lambda \mathbb{E}_{\hat{x}\sim P_{\hat{(x)}}}((||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2)$$

where the first part of the equation is the original Discriminator loss, and the second part is the so called gradient penalty term. The $\hat{x}$ is sampled from $\tilde{x}$ and x with t uniformly between 0 and 1:

$$\hat{x} = t\tilde{x} + (1 - t)x$$

with

$$0 \leq t \leq 1$$

2.2.1.2 Conditional GANs

In an unconditioned generative model, there is no control on modes of the data being generated. However, by conditioning the model on additional information it is possible to direct the data generation process. Such conditioning for example

could be based on class labels, or other information regarding the data. Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. y could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding y into both the discriminator and generator as an additional input layer.

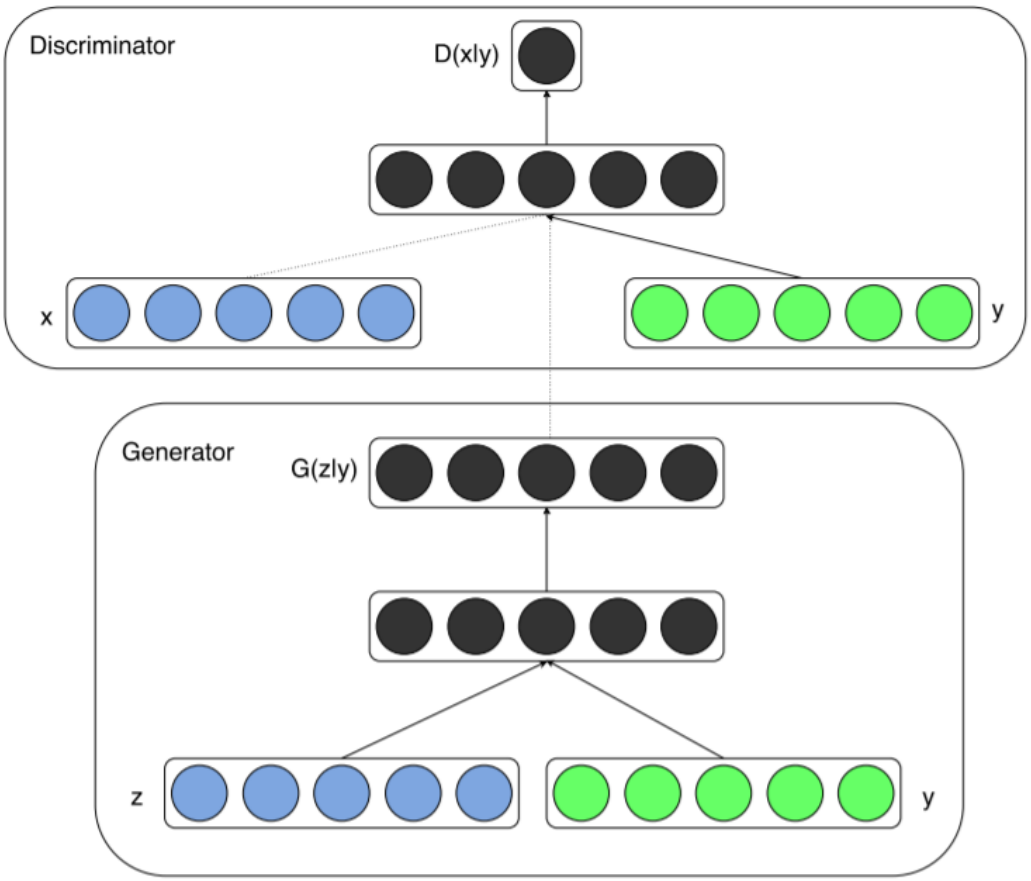


Figure 2.10. Schematic representation of a basic cGAN architecture [25].

2.2.2 Interpolation in GAN latent space

GANs exhibit two strong signs of generalization. First, the generator translates linear interpolations in the noise space into semantic interpolations in the sample feature space. A linear interpolation in the noise space will generate a smooth interpolation of realistic samples. Second, the generator allows linear arithmetic in the noise space. Similarly to word embeddings, linear arithmetic indicates that the generator organizes the noise space to disentangle the nonlinear factors of variation of natural images into linear statistics. This advantage of GANs allows them to analyze complex gene-gene relationships and gene expression data in pseudotime, between two real data points [29]. An example can be seen in image-image interpolation in Figure 2.11.

Figure 2.11. Illustration of interpolations obtained with a GAN model on the CelebA dataset. Each row corresponds to an image pair, and the leftmost and rightmost images are actual images from the training set. Given two images i and j, an interpolated latent vector z between z_i and z_j can be obtained and show the reconstruction $g(z)$ [29].

2.3 Quantification of multiple gene expression in individual cells

Cell behavior can be determined by analysing the associated gene expression patterns. For a long time these assessments could be performed only at cell population level due to technical limitations. Therefore, they determine the average gene expression within a population, overlooking possible cell-to-cell heterogene-

ity that could lead to different cell behaviors or cell fates. By accessing gene expression information on single cell level, one can understand more types of biological events and differentiation processes.

In the past few years new developments of single-cell RNA sequencing has deepened our understanding of the cell as a functional unit, as with new techniques gene expression profiles of up to a million of individual cells can be quantified, revealing new sub-populations of cells which were previously masked within analyses of gene expression performed on bulk cell populations. New technologies, such as *Fluidigm C1*, *DropSeq*, *Chromium 10X*, and *SCI-Seq* are able to characterize the transcriptional profile of hundreds up to many thousands of single cells at a time. All of the previously mentioned methods rely on labeling mRNA molecules with DNA barcodes during reverse transcription, which is used to trace the transcripts back to their cells of origin. The differences between them are how they manage to separate cells and label the mRNA molecules. However, they share the same computational pipeline to process and analyze the generated data [30], [31].

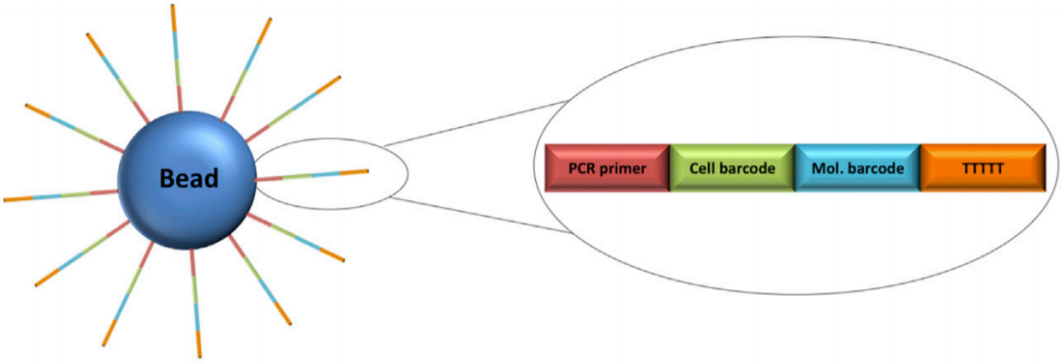


Figure 2.12. The structure of a DropSeq single-cell sequencing bead. The oligos extending from the bead have a PCR primer, a cell barcode that is unique to the bead to label each cell, a UMI that is unique to each individual oligo arm to allow unique labeling of each captured molecule, and a poly(T) tail to capture poly(A)-tailed mRNAs. The image was taken from [30].

2.4 Analysis of Single-Cell RNA-Sequencing Data

2.4.1 Dimensionality reduction methods

Dimensionality reduction techniques have been pivotal in enabling researchers to visualize high-dimensional data. Although principal component analysis (PCA) has historically been the most commonly used method for dimensionality reduction, the importance of nonlinear dimensionality reduction techniques has recently been recognized. Nonlinear dimensionality reduction methods include Isomap2, Diffusion Map and t-distributed stochastic neighborhood embedding (t-SNE, renamed viSNE). t-SNE is currently the most commonly used technique in single-cell analysis. It has been used to efficiently reveal local data structure and is widely used to identify distinct cell populations in cytometry and transcriptomic data. However, t-SNE suffers from limitations such as loss of large-scale

information (the intercluster relationships), slow computation time and inability to meaningfully represent very large datasets. Novel methods, like Uniform Manifold Approximation and Projection (UMAP) are recently introduced, and they can solve the shortcomings of the older methods. However, they are still in the experimental phase [32].

2.4.2 Theoretical basis for Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP) is a novel non-linear dimensionality reduction technique and is constructed from a theoretical framework based in Riemannian geometry and algebraic topology. UMAP appears to have some significant advantages over t-SNE: It's faster than t-SNE, it captures global structure better than t-SNE, and UMAP is also a general-purpose dimensionality reduction technique that can be used as preprocessing for machine learning. UMAP also has a solid theoretical backing as a manifold approximation technique, whereas t-SNE is primarily a visualization heuristic.

The UMAP algorithm assumes that the data points of a high-dimensional data set lie on a lower dimensional manifold. A manifold is an object that is embedded in some higher dimensional space. To construct topological spaces and manifolds, one can use simplical complexes, which allow one to reduce the complexities of dealing with the continuous geometry of topological spaces to the task of relatively simple combinatorics and counting. The simplest building blocks to construct a k-dimensional object are called **simplices**. A k-dimensional simplex is called a k-simplex, and they are made out of taking the convex hull of $k + 1$ independent points. Examples for the first four simplices can be seen in Figure 2.13.

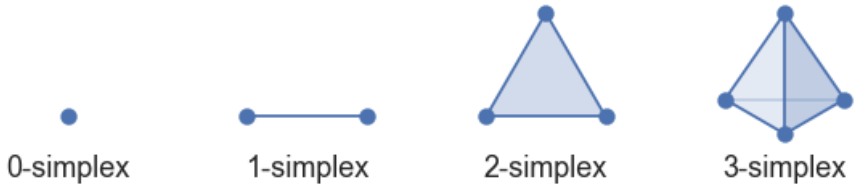


Figure 2.13. Low dimensional simplices [33].

Simplices can be used as building blocks for more complex constructs by gluing together, to create a so called **simplical complex**. A simplicial complex κ is a set of simplices such that any face of any simplex in κ is also in κ (ensuring all faces exist), and the intersection of any two simplices in κ is a face of both simplices. This can be imagined informally as the simplices are glued together along with their faces.

To apply these theoretical tools from topology to finite sets of data points, one needs to be able to construct simplical complexes from a topological space. This can be done by constructing a Čech complex given an open cover (a family of sets whose union is the whole space) of a topological space. Constructing a Čech complex is relatively simple: let each set in the cover be a 0-simplex; create a 1-simplex between two such sets if they have a non-empty intersection; create a 2-simplex between three such sets if the triple intersection of all three is non-empty; and so on. This way the background topological theory (Nerve theorem) provides guarantees about how well this simple process can produce something that represents the topological space itself in a meaningful way.

For finite set of data samples we assume that the data samples are drawn from some underlying topological space, and then we generate an open cover of it. To approximate the open cover, one can simply create spheres with a fixed radius at each data point. This way the Čech complex associated with the cover will have

a 0-simplex for each data point.

Below is a simple example of the whole process with 2 dimensional data, where the underlying manifold is a sine wave (Figure 2.14).

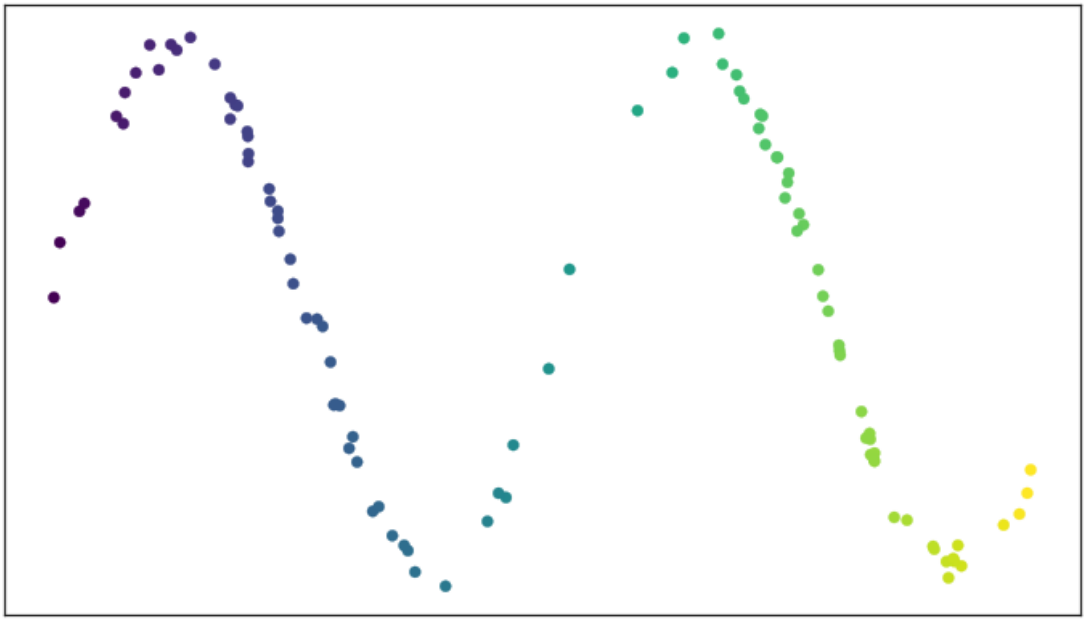


Figure 2.14. Example dataset of a noisy sine wave [33].

In two dimensions, we can visualize the open cover as circles (Figure 2.15).

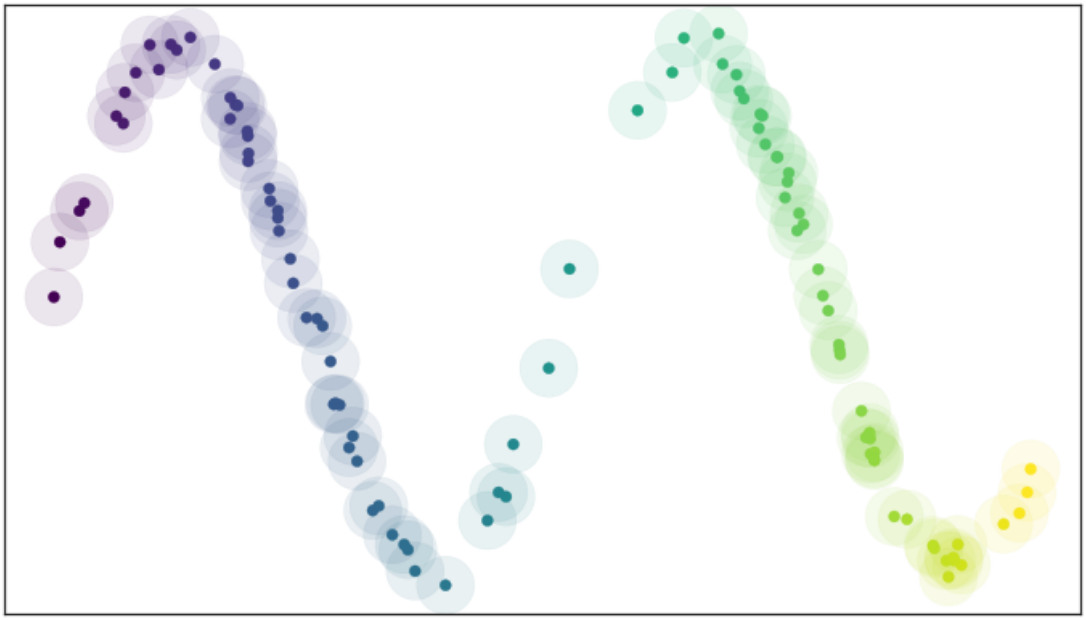


Figure 2.15. Open cover of the example data [33].

By constructing the simplical complex as discussed above, we can visualize the simple simplexes (0, 1, 2 dimensions) as points, lines and triangles. For topological data analysis it is more computationally feasible to construct only a so called Vietoris–Rips complex, which is similar to a Čech complex, but is entirely determined by the 0- and 1-simplices. This way the topological representation of the data is a graph, and finding a low dimensional representation can be described as a graph layout problem.

The problem with this approach is that choosing the right radius for the balls that make up the open cover is hard. For a too small value, the simplical complex may split up, for too big value the underlying manifold structure could vanish. The root of the problem lies in the fact, that the Nerve theorem, which provides justification that this process captures the topology, assumes uniformly distributed data on the manifold. If we consider data that is uniformly distributed along the same manifold it is relatively easy to pick a good radius (a little above half the

average distance between points), as can be seen in Figure 2.16.

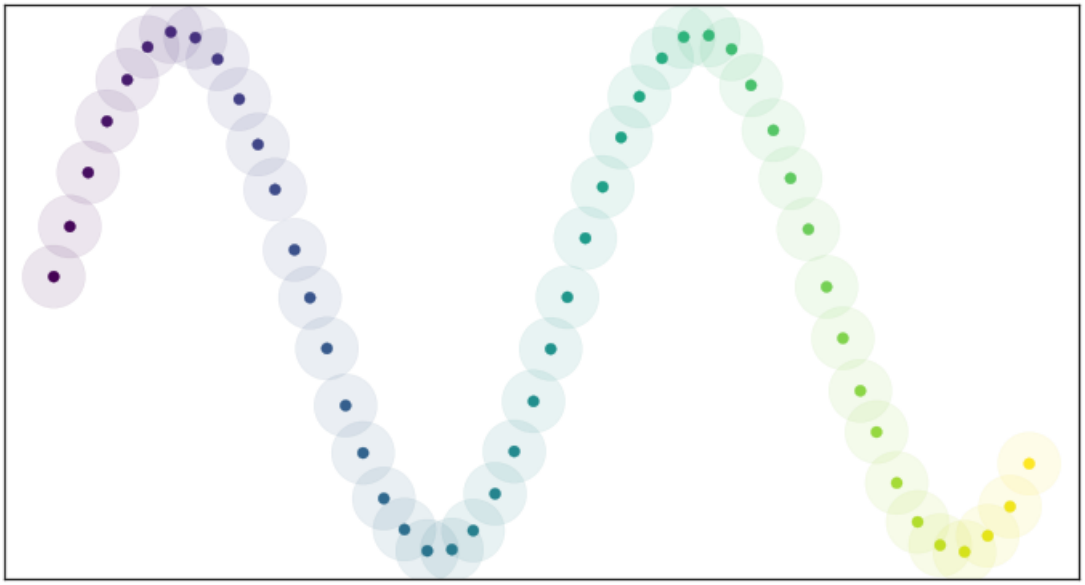


Figure 2.16. Open circles over uniformly distributed data [33].

However, real world data is rarely uniformly distributed on the manifold. To overcome this issue, we utilize the following assumption: we assume the data is uniformly distributed on the manifold, and the notion of distance is varying across the manifold, the space itself is stretching or shrinking according to where the data appear sparser or denser. Following the computations by McInnes et al.[34] the results are that a unit circle about a point stretches to the k-th nearest neighbor of the point, where k is the sample size we are using to approximate the local sense of distance. This way each point is given its own unique distance function, and we can simply select circles of radius one with respect to that local distance function. The new unit circles can be seen in Figure 2.16.

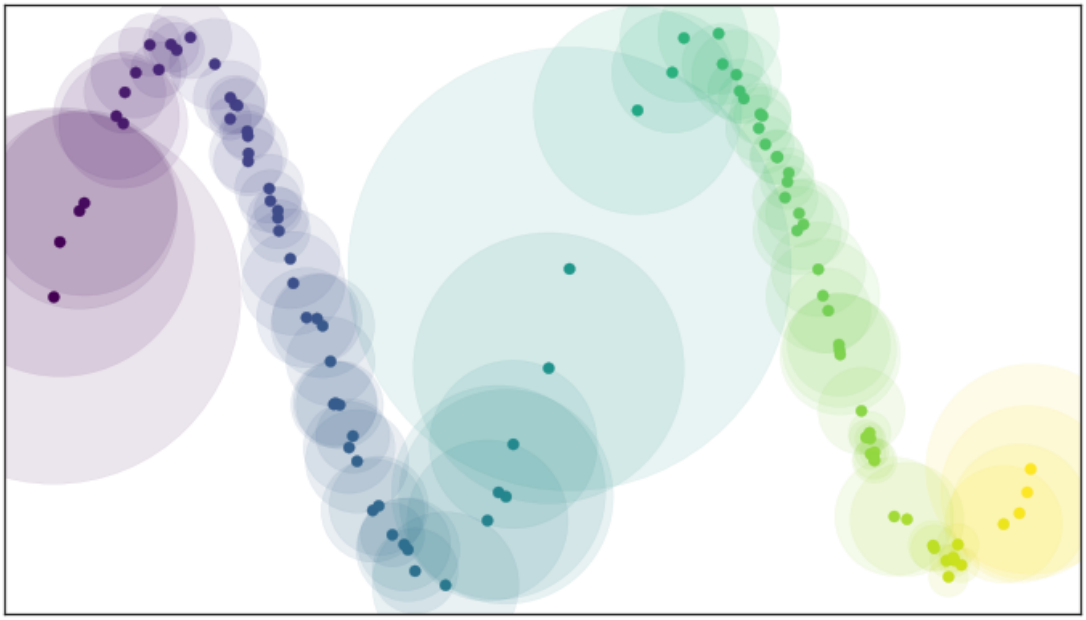


Figure 2.17. Open circles of radius one with a locally varying metric [33].

This way the radius parameter for the circles are exchanged to the k parameter of the k-th nearest neighbor, but choosing the correct k-value is less dataset dependent and in practice UMAP uses the efficient Nearest-Neighbor-Descent algorithm of Dong et al. The other benefit of this Riemmanian metric based approach is that every point has a local metric space associated to it, thus we could measure how far the points are from each other (respect to the local metric). This way one can introduce fuzzy topology, instead of the binary yes-or-no values. The simple visualization of these fuzzy open circles can be seen in Figure 2.18.

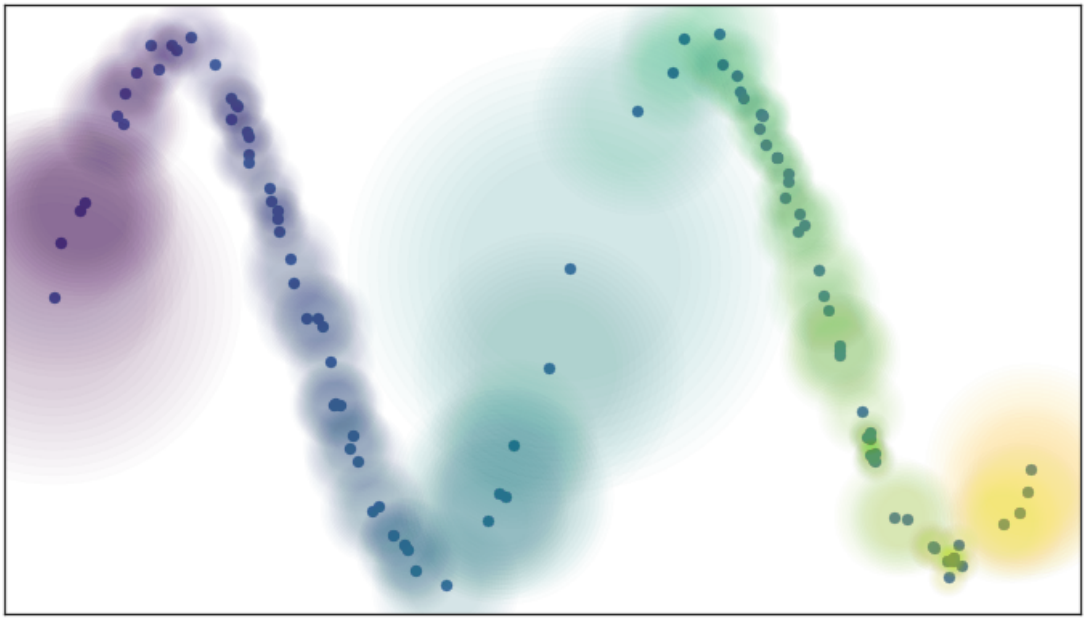


Figure 2.18. Fuzzy open circles of radius one with a locally varying metric [33].

In real world data, especially in higher dimensions a new problem arises with this approach: a lot of points can become totally isolated. To overcome this, one can add the idea of local connectivity, which means no point should be completely isolated - it should connect to at least one other point. In terms of fuzzy open sets what this amounts to is that we should have complete confidence that the open set extends as far as the closest neighbor of each point. The implementation of this solution is that having the fuzzy confidence decay in terms of distance beyond the first nearest neighbor, which can be observed in Figure 2.19.

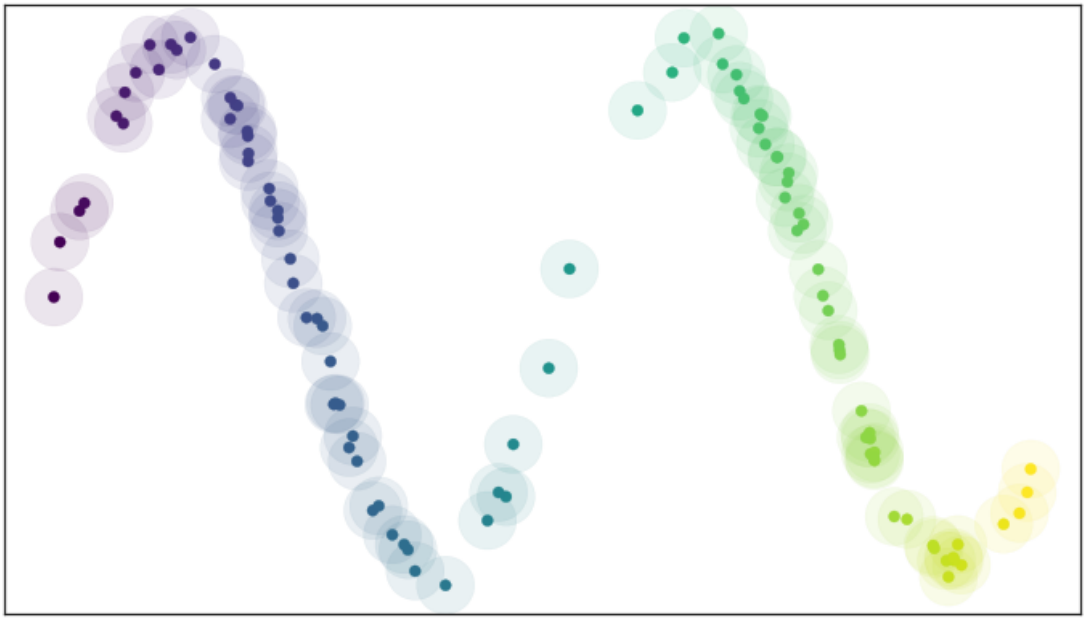


Figure 2.19. Local connectivity and fuzzy open sets [33].

By defining local metrics to each data point, one can end up incompatible metrics between them, for example the distance between data point a and data point b might be 0.6, but between b and a, it could be 1.5. To overcome this problem, one can simply take the union of the fuzzy simplical sets. In graph terms what we get is the following: if we want to merge together two disagreeing edges with weight a and b then we should have a single edge with combined weight $a + b - ab$. By applying this process to union together all the fuzzy simplicial sets we end up with a single fuzzy simplicial complex, which we can think of as a weighted graph as seen in Figure 2.20.

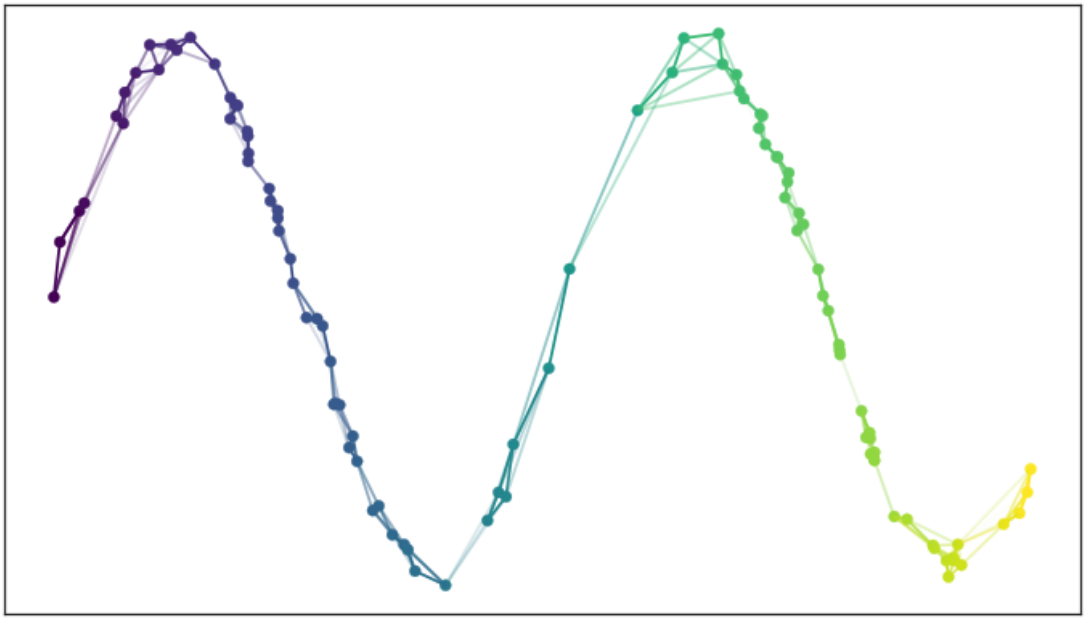


Figure 2.20. Graph with combined edge weights [33].

2.4.3 Finding a low dimensional representation with UMAP

When finding a lower dimensional representation of our data, we would like to ensure that this representation has a similar fuzzy topological structure as the original one. To achieve this we can use the same procedure as before, however we want the data to be on a particular manifold: the low dimensional Euclidean space we are trying to embed in and we explicitly want the distance on the manifold to be standard Euclidean distance with respect to the global coordinate system, not a varying metric.

To compare how good the lower dimensional representation to the higher one, one can compare the graph representation of the two: calculate if the same edges are present between the data points. This can be done, by calculating the cross-entropy of the weights, represented as vectors.

$$H = \sum_{e \in E} w_h(e) log\left(\frac{w_h(e)}{w_l(e)}\right) + ()1 - w_h(e)) log\left(\frac{1 - w_h(e)}{1 - w_l(e)}\right)$$

The first term of the equation creates an attractive force between the points e spans whenever there is a large weight associated with the high dimensional case. The second term creates a repulsive force between the ends of e whenever $w_h(e)$ is small. On balance this process of pull and push, mediated by the weights on edges of the topological representation of the high dimensional data, will let the low dimensional representation settle into a state that relatively accurately represents the overall topology of the source data.

During the optimization of the low dimensional embedding, UMAP uses stochastic gradient descent, making use of probabilistic edge sampling and negative sampling, described by Dong et al.[35]. Finally since the Laplacian of the topological representation is an approximation of the Laplace-Beltrami operator of the manifold we can use spectral embedding techniques to initialize the low dimensional representation into a good state.

By comparing the UMAP algorithm to other dimensionality reduction methods on various benchmark dataset, one can see UMAP scales better than t-SNE (Figure 2.21).

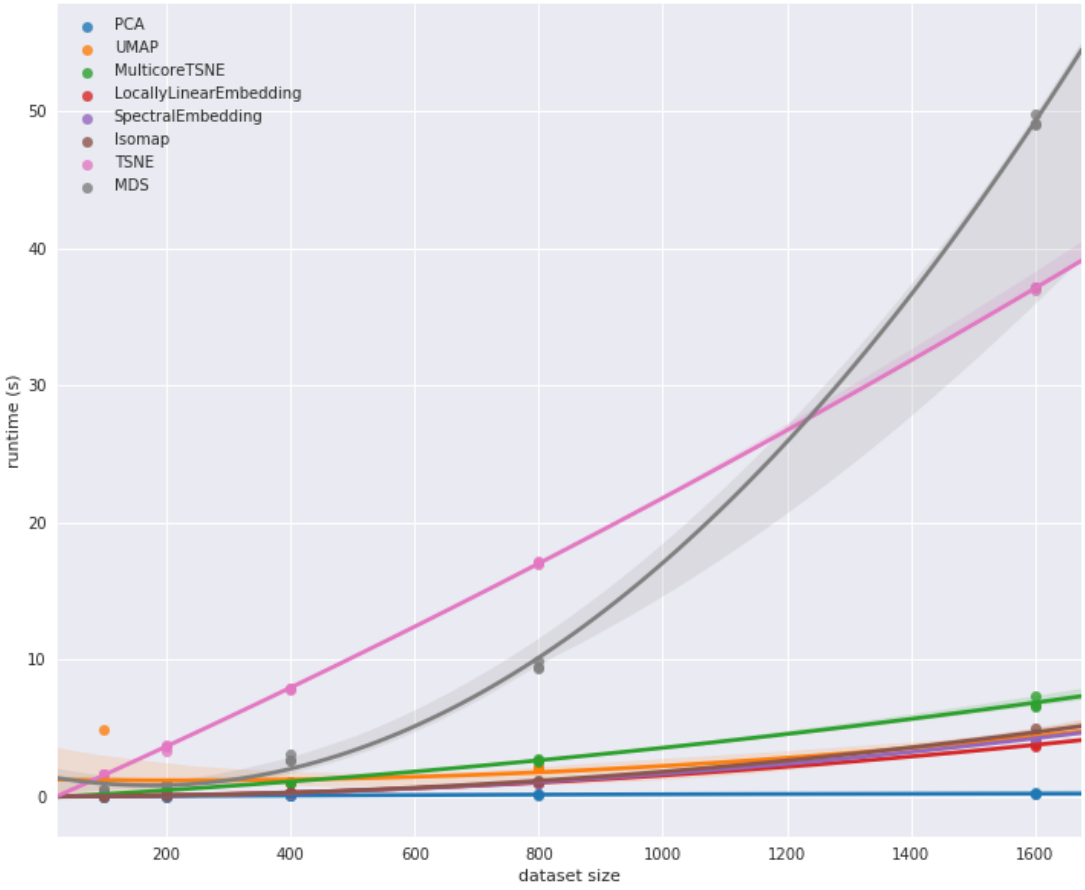


Figure 2.21. Comparing the scalability of the various dimensionality reduction methods. Image taken from [34].

2.4.4 Splatter

Splatter, is an R package for reproducible and accurate simulation of single-cell RNA sequencing data. It is a framework designed to provide a consistent interface to multiple published simulations and to make comparisons between simulated and real data [36].

Splatter implements six different simulation models, from the simplest with the least assumptions about the data, to the most complex one. The Splatter simulation process consists of two steps. The first step estimates the parameters

required for the simulation from a real dataset. In the second step, it uses the estimated parameters to generate a synthetic scRNA-seq dataset using the selected models. The main result of the simulation step is a matrix of counts. Next, I discuss two particular models further, used for the comparison with the synthetic data generated by the scsGAN.

2.4.4.1 Simple

The most basic model is called **Simple**, and it uses gamma distribution to simulate mean expression levels for each gene, and the negative binomial distribution to generate a count for each cell based on these means, with a fixed dispersion parameter. It should be noted, that this model is primarily included as a baseline reference and is not intended to accurately reproduce many of the features of scRNA-seq data.

2.4.4.2 Splat

The **Splat** model has been developed to capture many features observed in real scRNA-Seq data, including high expression outlier genes, differing sequencing depths (library sizes) between cells, trended gene-wise dispersion, and zero-inflation. The core of the Splat simulation is the gamma-Poisson hierarchical model where the mean expression level for each gene is simulated from a gamma distribution and the count for each cell is subsequently sampled from a Poisson distribution, with modifications to include expression outliers and to enforce a mean-variance trend.

More specifically, the Splat simulation initially samples gene means from a Gamma distribution. To capture outliers even more accurately a probability that a gene is a high expression outlier can be specified. Splatter then add these outliers to the simulation by replacing the previously simulated mean with the median of

the simulated gene means multiplied by an inflation factor. The inflation factor is sampled from a log-normal distribution.

The library size (which is the total number of counts) varies within an scRNA-seq experiment and Splatter models this library size using a log-normal distribution and uses the simulated library sizes to proportionally adjust the gene means for each cell. This allows Splatter to alter the number of counts per cell independently of the underlying gene expression levels.

It is also known that there is a strong mean-variance trend in RNA-Seq data, where lowly expressed genes are more variable and highly expressed genes are more consistent. Splat enforce this trend by simulating the biological coefficient of variation for each gene from a scaled inverse chi-squared distribution, where the scaling factor is a function of the gene mean and then apply further transformations on it.

The high proportion of zeros is another key feature of scRNA-seq data, and one cause of which is the technical dropout. Splatter uses the relationship between the mean expression of a gene and the proportion of zero counts in that gene to model this process and use a logistic function to produce a probability that a count should be zero. The probability of a zero for each gene is then used to randomly replace some of the simulated counts with zeros using a Bernoulli distribution.

The detailed process how Splat models a distribution can be seen in Figure 2.22.

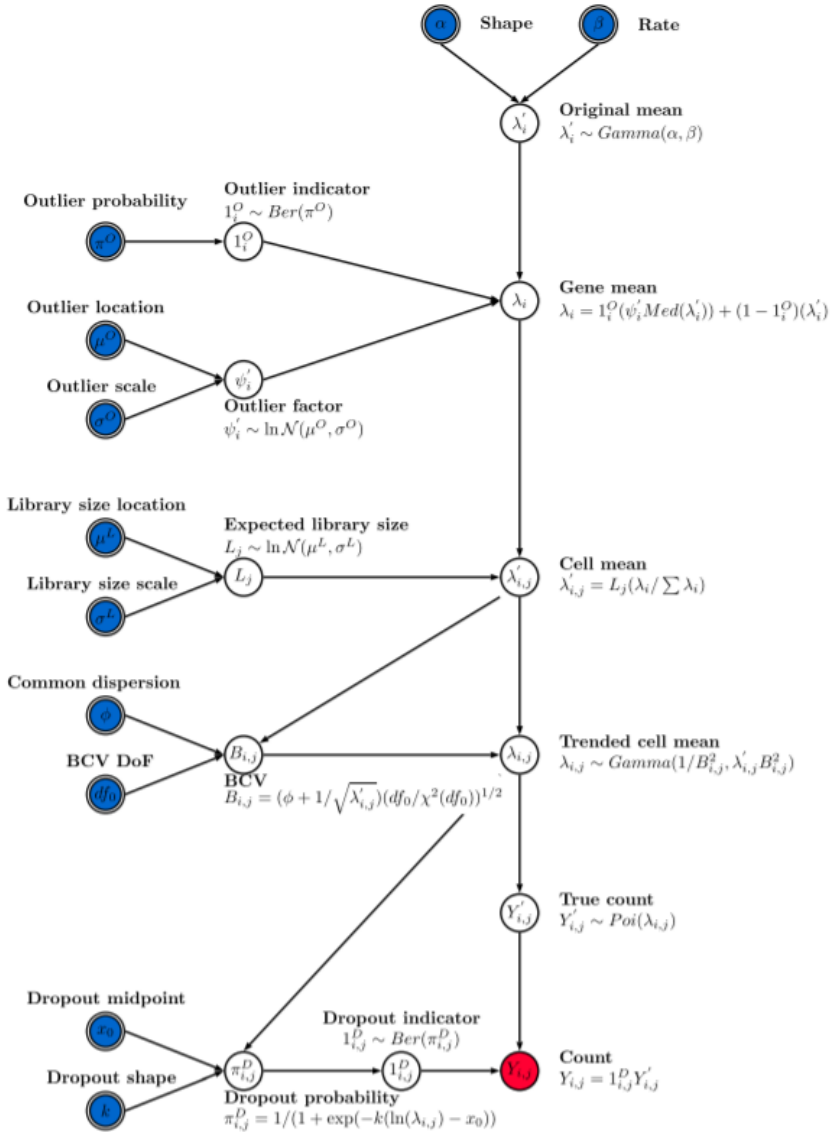


Figure 2.22. The Splat simulation model. Input parameters are indicated with double borders and those that can be estimated from real data are shaded blue. Red shading indicates the final output [36].

2.4.5 Monocle

Monocle is an analysis toolkit for single-cell RNA-Seq experiments, using the R statistical computing environment. Monocle performs differential expression and

time-series analysis for single-cell expression experiments and it orders individual cells according to progress through a biological process, without knowing ahead of time which genes define progress through that process. Monocle also performs differential expression analysis, clustering, visualization, and other useful tasks on single cell expression data. It is designed to work with RNA-Seq and qPCR data, but could be used with other types as well. To compare the results obtained by the scsGAN framework, especially the latent space interpolation experiments, Monocle is an important benchmarking tool. In contrast to other methods, Monocle uses the so called DDRTree algorithm for learning principal graphs to reconstruct single-cell trajectories [37].

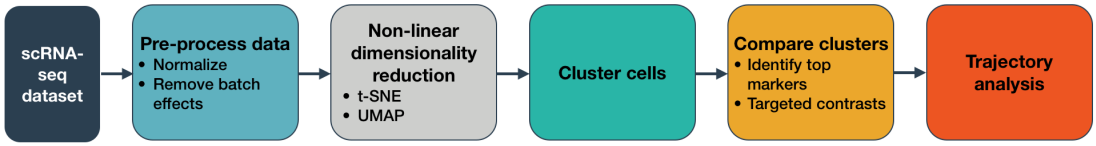


Figure 2.23. The Monocle 3.0 workflow [37].

Chapter 3

The scsGAN model

This chapter introduces the proposed scsGAN model, and the rest of the thesis describes how it was implemented, trained, and performed on various datasets. The implementation, training, and generation of the results were all my own work.

Overview of the model

The scsGAN model consists of two neural networks, implemented using the **Tensorflow** open source library. Both the Generator and the Discriminator models are fully connected neural networks with two hidden layers. The generator network has an input layer of n neurons, which corresponds to the number of input features from the data set. The final layer of the discriminator network lacks an activation function, in line with other Wasserstein GANs using a gradient penalty loss function (L2 regularisation scale = 10).

I used a Leaky Rectified Linear Unit (LReLU) with a coefficient of 0.2 as the activation function for both neural networks. The neural networks were trained by backpropagation and RMSProp was utilized with the learning rate of $5e-5$.

Table 3.1. Layers of the Generator network without the input and output layers.

Layer number	Layer type	Number of neurons	Activation function used
1	Dense	600	Leaky RELU
2	Dense	600	Leaky RELU

Table 3.2. Layers of the Discriminator network without the input and output layers.

Layer number	Layer type	Number of neurons	Activation function used
1	Dense	600	Leaky RELU
2	Dense	600	Leaky RELU

The batch size was set to 32.

In this work the latent space of the generator network consists of a 100-dimensional additive Poisson and Gaussian distributed latent variable z where $z = N(\mu = 0, \sigma = 0.1 \cdot max) + P(\lambda)$, max is the largest value in the data set to simulate the Poisson distributed nature of scRNA-seq count data.

3.1 Training of the model

As discussed above, the Generator and the Discriminator networks were trained simultaneously, and the Discriminator was updated 5 times for every update of the Generator network for better convergence.

There is currently no standard method for assessing generative models of gene expression data. The output of the GAN was computationally validated by examining the followings during the training and validation:

- Correlation between the genes of the generated samples and unseen samples
- Pairwise correlation between genes in the generated and unseen samples
- Discriminator loss

- Plotting the generated samples with UMAP on the unseen samples

The training is considered finished when these metrics converge into stable values. Together these results demonstrate that the Generator network is not memorising and reproducing training samples but is instead inferring relationships between gene expression values in order to output convincing heterogeneous generated cells.

Chapter 4

Methods

4.1 Overview of the datasets

Various datasets with different cell numbers and species were selected to test the capabilities of the proposed scsGAN method, and they are summarized in Table 4.1. To better visualize how certain aspects of the model works, like the latent space interpolation, an image-type datasets were included featuring astronomical pictures. Regarding the scRNA-seq datasets, the same transformations and data pre-processing steps were applied as it was described in their corresponding article.

Table 4.1. Description of the datasets used in the thesis.

Name	Reference	Species	Number of samples	Short description
Tung	[38]	Human	564	Induced pluripotent stem cells
Engel	[39]	Mouse	203	Natural killer T cells
PBMC	[40]	Human	16893	Peripheral blood mononuclear cells
Joost	[41]	Human	1422	Eleven epidermal cell types
Galaxy Zoo	[42]	-	61578	JPG images of 61578 galaxies

4.2 Approximating scRNA-seq distributions

According to the Splatter paper [36] existing scRNA-seq analysis packages, and any new methods that are being developed, like the scsGAN model, should demonstrate two properties:

- Prove that they can do what they claim to do, whether that is clustering, lineage tracing, differential expression testing or improved performance compared to other methods.
- They can produce some meaningful biological insight. However, it is specific to particular studies.

Zappia et al. showed how Splatter compares to previously published simulations based on real datasets.
Following their workflow, I read in each of the datasets, filtered the top 5000 most variable genes, and randomly sampled 200 cells to use for the estimation step and each simulation was consisted of 200 newly generated cells. I visualized with UMAP the generated and the validation original cells, which were drawn from the original data, but were not included in the training step. I also calculated the pairwise Pearson correlation coefficient between the genes of the original validation set and the generated cells.

4.3 Interpolation in GAN latent space

4.3.1 Retrieving latent space vector

To retrieve the latent space vector z from an arbitrary gene expression profile x such that $G(z) = x$, I randomly generated 10000 cell expression profiles and saved the corresponding latent space vector for each of them. Finally, I selected

the expression profile which has the smallest mean squared error between the expression profile of the real sample and the generated ones.

4.3.2 Latent space arithmetic for interpolation

To interpolate between two real data points first I calculated the two latent space vectors of the real expression profiles using the method described above. To obtain simulated gene expression time series data I interpolated n points between z_1 and z_2 and generated n intermediate gene expression profiles from these, where n is the number of points to be interpolated.

4.4 Out of bag style transfer using latent space arithmetics

In other fields where GANs have been applied, like image generation vector arithmetic in the latent space leads to meaningful outputs. Lotfollahi et al. [40] showed the same meaningful outputs can be generated using scRNA-seq data by obtaining so called **perturbation vectors**, which then can be applied to arbitrary expression vectors, to perturbate the corresponding cellular states. This method can be used to learn these perturbation vectors from the training set of unperturbated and perturbated cells, and then apply them on unseen cells.
In the simplest case of a binary perturbation event let every cell i with expression profile x_i be characterized by a variable p_i, which represents a discrete attribute across the whole manifold, such as perturbation, species or batch. Let us further consider the conditional distribution $P(x_i|z_i, p_i)$, which assumes that each cell x_i comes from a low-dimensional latent space representation z_i in condition p_i . I use a Wasserstein GAN to model $P(x_i|z_i, p_i)$ in its dependence on z_i and vector arithmetics in the latent space of the GAN to model the dependence on p_i.

In this example, the model is only trained on the unperturbated cells, and then it predicts the latent space representation of perturbated cells of a given cell type A using:

$$z_{i,A,p=1} = z_{i,A,p=0} + \delta$$

where $z_{i,A,p=0}$ and $z_{i,A,p=0}$ denote the latent representation of cells with cell type A in conditions $P = 0$ and $P = 1$, respectively, and δ, is the difference vector of medians between cells in the training set in condition 0 and 1. In the current scsGAN implementation 50 cells were selected from each group from each condition in the train set, and the median perturbation latent vector was calculated using those. The process is demonstrated in Figure 4.1, where the original method proposed by using a VAE is shown (the similar workflow applies with GANs too, without the encoder part).

As Lotfollahi et al. [40] demonstrated, I applied the scsGAN model to a published human peripheral blood mononuclear cells (PBMCs) dataset, which were stimulated with interferon (IFN-β). I followed the same pre-processing steps, and evaluated the results using hierarchical clustering of the mean of gene expression profiles in every group of cells with each condition including the generated cells after the perturbation vector was applied. The results were also visualized using the UMAP dimensional reduction method, with arrows overlaying the plot showing where the original, and the generated cells were embedded.

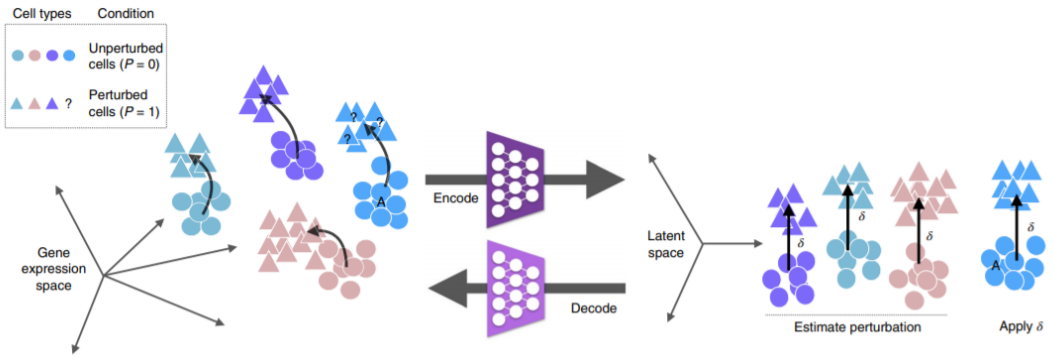


Figure 4.1. Training and predictions in gene expression and the latent space. [40].

4.5 Data augmentation using scsGAN

Data Augmentation encompasses a suite of techniques that enhance the size and quality of training datasets such that better Deep Learning models can be built using them. Generally, to build useful Deep Learning models, the validation error must continue to decrease with the training error and data augmentation is a very powerful method of achieving this. Data augmentation approaches are basically overfitting the training dataset. This is done under the assumption that more information can be extracted from the original dataset through augmentations. These augmentations artificially inflate the training dataset size by either data warping or oversampling.

The use of GANs in imaging, mostly medical imaging is well documented such as CT denoising, accelerated magnetic resonance imaging, PET denoising, and the application of super-resolution GANs in retinal vasculature segmentation. However, using GANs for data augmentation for scRNA-seq datasets are less explored [43].

To validate the scsGAN model can be used for data augmentation I trained the model on the PBMC dataset with all cell groups and conditions, and:

- Trained and predicted with a Random Forest algorithm with default hyperparameters on the cell group target variable, using a 70-30% train-test split.

- Generated 50-50 cells for each group, by selecting 50-50 real cells belonging that particular group and searched for the closest latent space representation to it, as discussed before. Trained and predicted with a Random Forest as before, and compared the two accuracies of the predictions on the test set.

4.6 Dimensionality reduction using scsGAN

To be able to discriminate successfully between the generated and real samples, the Discriminator must discard uninformative genes. To test this hypothesis dimensionality reduction on the output of the second layer of the Discriminator network with PCA, t-SNE, UMAP were performed and compared.
The dimensionality reduction attributes of the scsGAN model have been tested on the PBMC dataset, with all of the conditions and cell groups included. The UMAP method was applied with default parameters on the first 100 principal components of the gene expression data, and also on the output of the second layer of the Discriminator network.

Chapter 5

Results

In this chapter, I will show the results obtained with the scsGAN after trained on various datasets. First, I show how the GAN approach is better, than the traditional algorithms on approximating scRNA-seq data, then I perform interpolation in the latent space. Next, I show the results of the out of bag style transfer, the GAN dimensionality reduction method, and also the possible data augmentation opportunities.

5.1 Approximating scRNA-seq distributions

The approximation of scRNA-seq data was performed on various datasets, and the results of two of these experiments (Tung, and Engel) are shown on the next figures.

In Figure 5.1 one can see how the loss of the Generator and the Discriminator changes in the epochs. As a short transition period, the loss of both of the models are saturated.

In Figure 5.2 one can see how the data points generated by the GAN (orange) overlap with the data points of the Tung dataset from the test set. We can

interpret this visually as the GAN has learned the underlying distribution well enough. This hypothesis is further strengthened by the pairwise cross correlation plot in Figure 5.3, which shows how the algorithm learned the corresponding gene-gene correlations in the data.

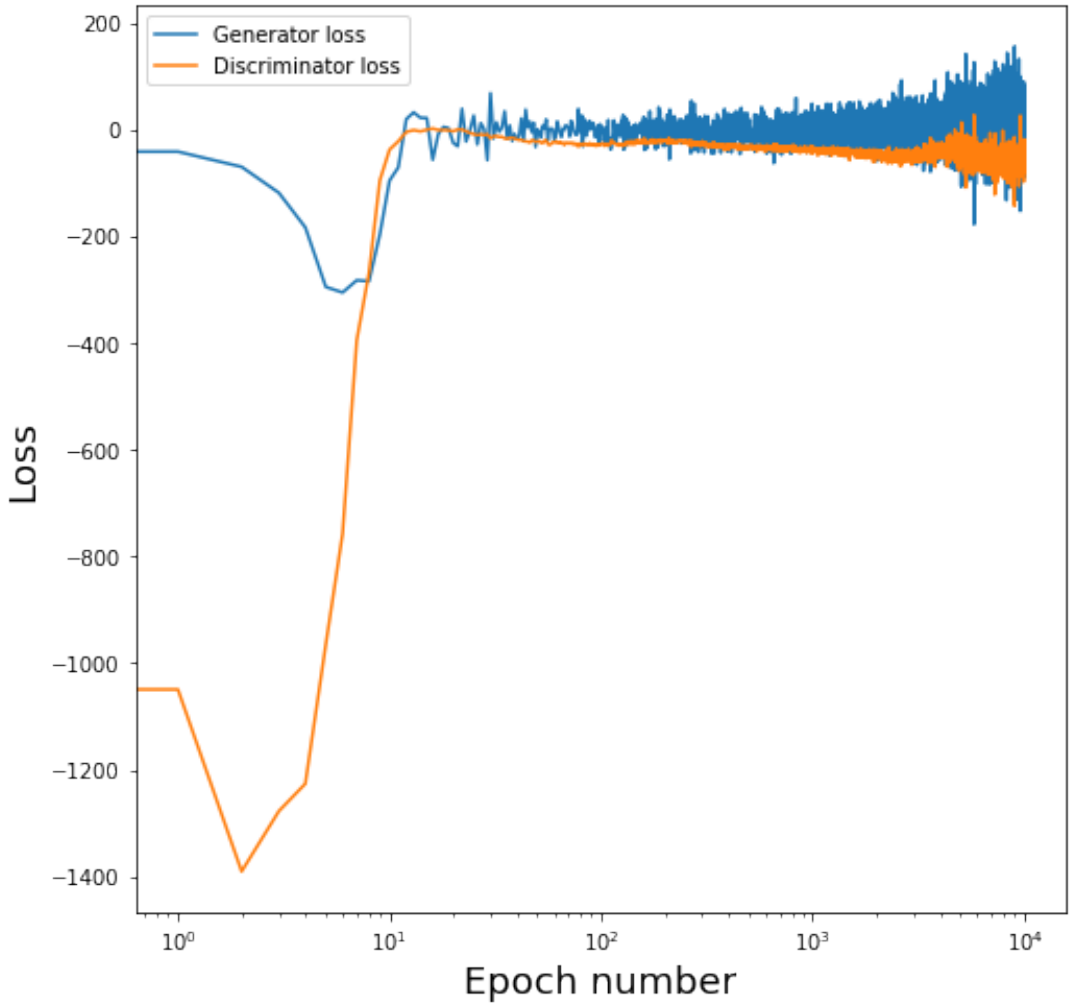


Figure 5.1. The loss of the Generator and the Discriminator models.

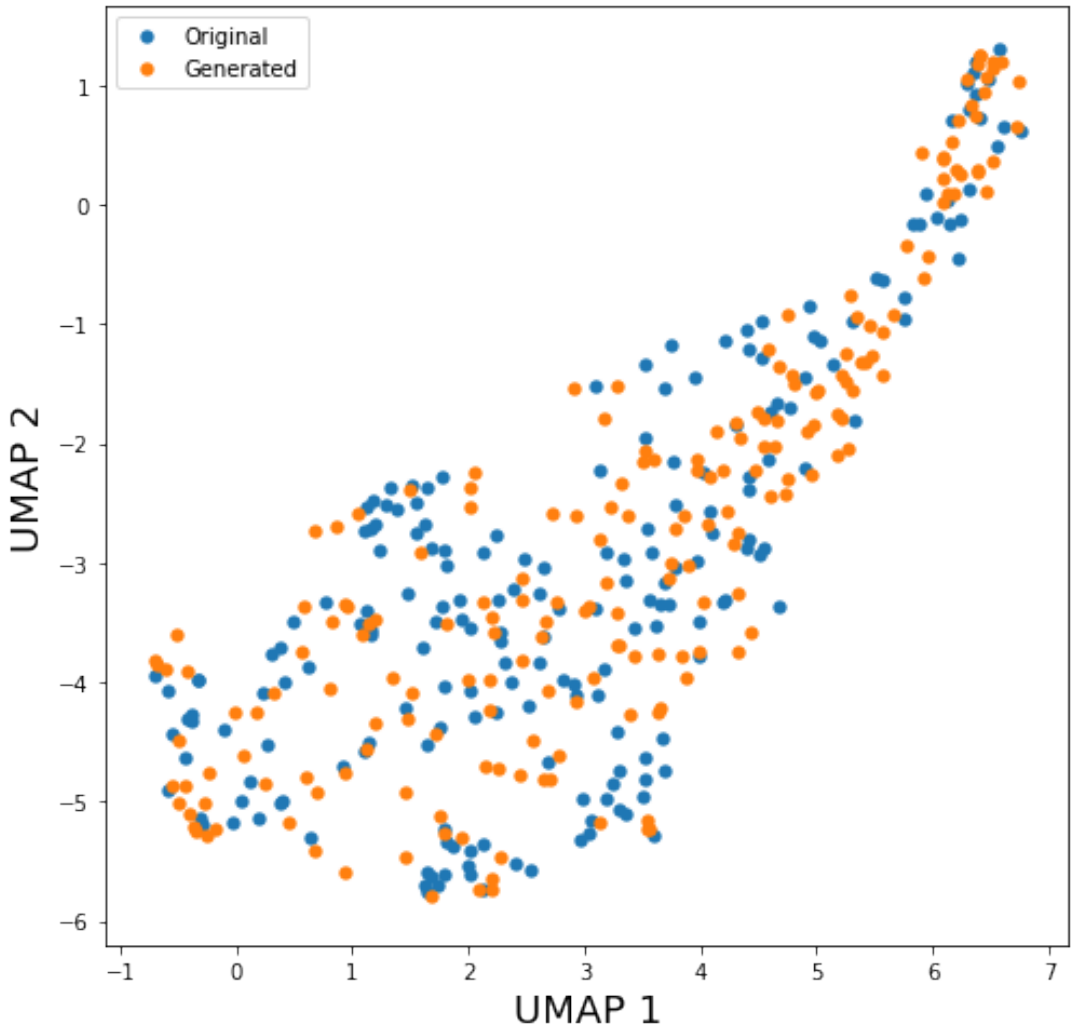


Figure 5.2. UMAP representation of the Tung dataset, and the generated cells.

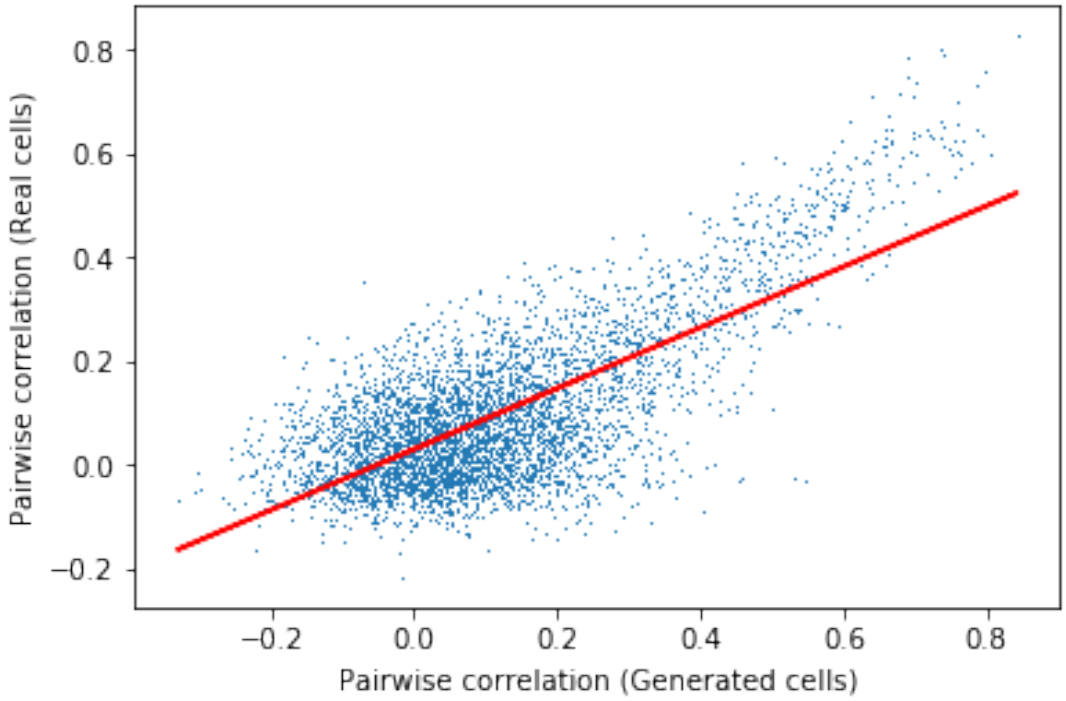


Figure 5.3. Visualizing the pairwise correlation coefficients of the genes across the Tung dataset and the generated cells.

Similar conclusions can be drawn from Figure 5.4 and Figure 5.5, which show the same analysis on the Engel dataset.

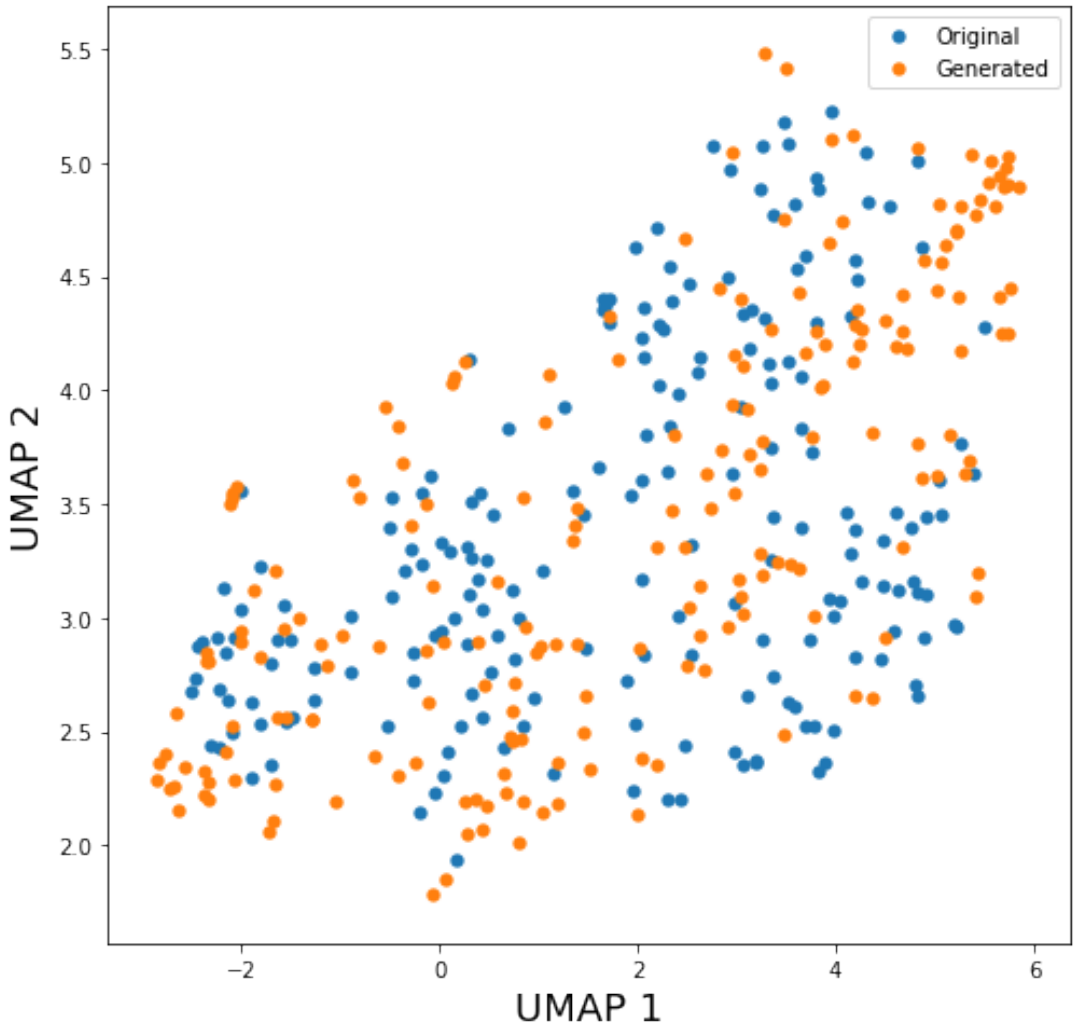


Figure 5.4. UMAP representation of the Engel dataset, and the generated cells.

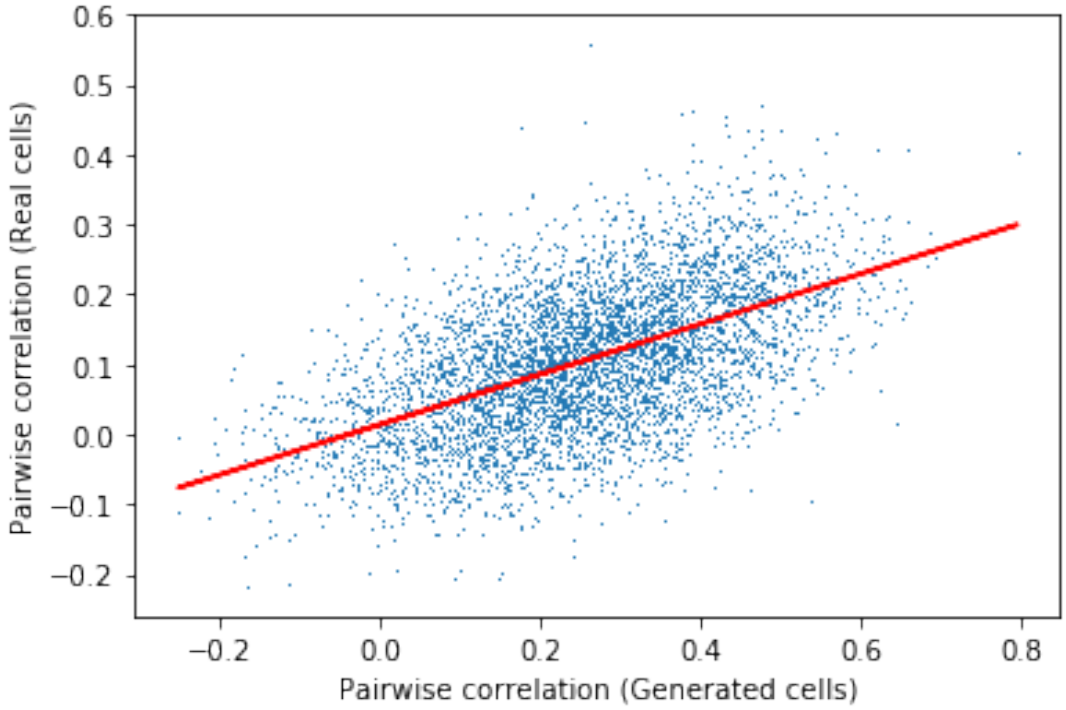


Figure 5.5. Visualizing the pairwise correlation coefficients of the genes across the Engel dataset and the generated cells.

A comprehensive analysis between the traditional algorithmic methods discussed in the previous chapters, and the GAN approach on the Tung dataset can be seen in Figure 5.6. The **Gan** name refers to the scsGAN method, and the **Orig** for the whole training dataset. One can see that except for the library size statistics, the scsGAN method performs the best everywhere. Regarding the library size, the GAN generates some cells, which are too big, containing an unreal amount of RNAs.

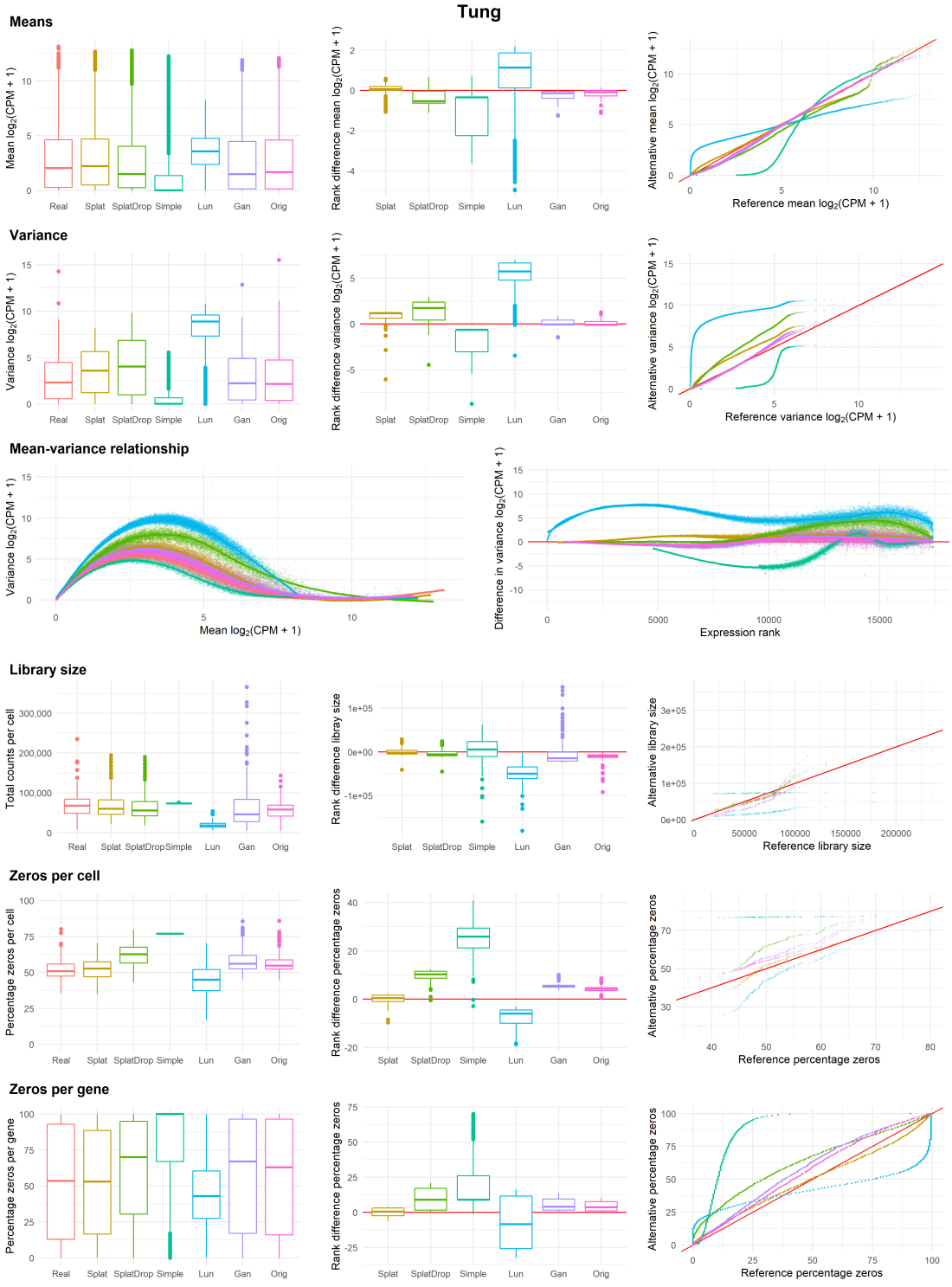


Figure 5.6. A comprehensive comparison of the traditional approximation methods and the results of the scsGAN.

5.2 Data augmentation

The results of the data augmentation task were promising, as one can see in Figure 5.7 the Generator generated samples that are grouped together by cell group labels.

The Random Forest algorithm trained on the cell group labels achieved 0.786% accuracy on the real samples, while achieving 0.580% accuracy on the pure synthetic samples meaning the Generator learned meaningful features from the data. The augmented real dataset mixed with synthetic samples resulted in the same 0.786% accuracy, which is not an improvement. The data augmentation part of the scsGAN method needs to be further explored, to discover the true possibilities of this approach.

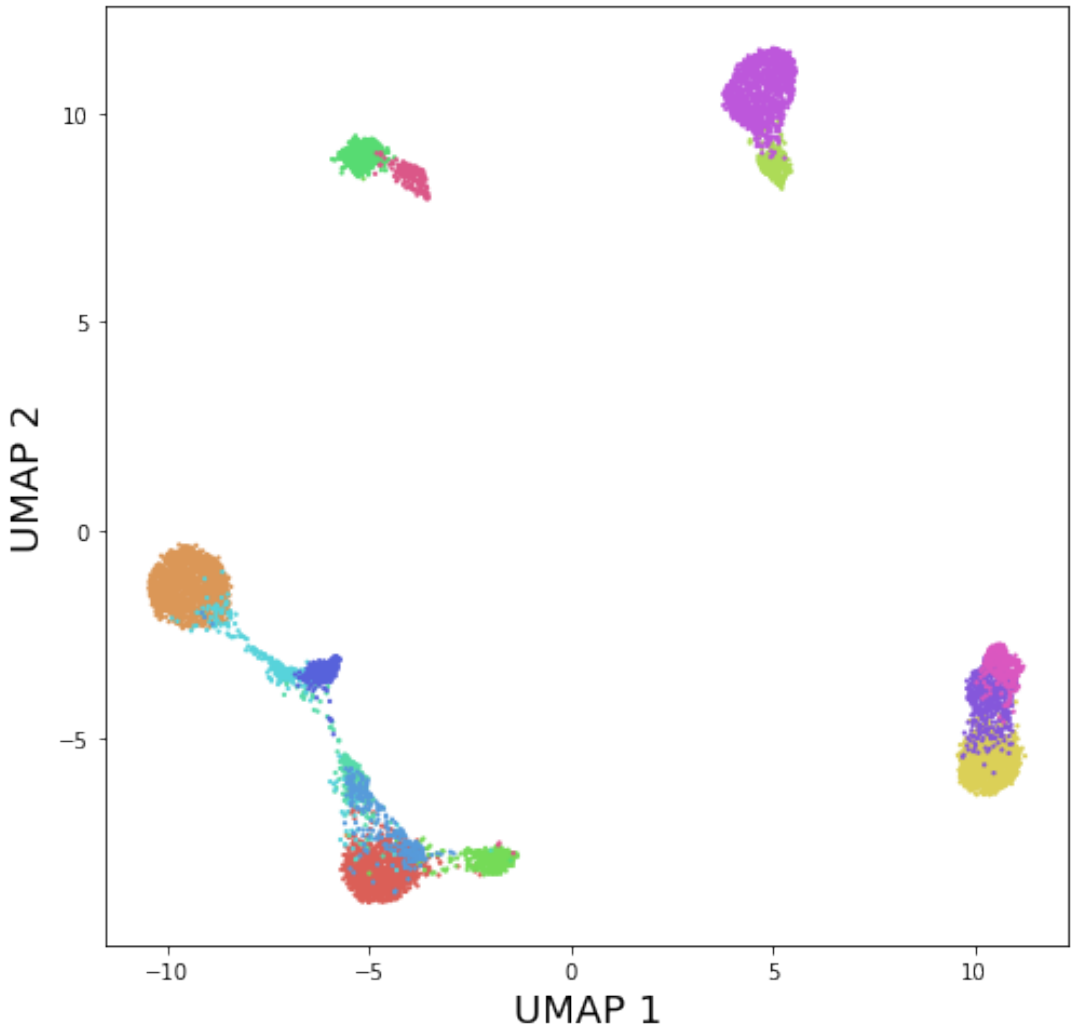


Figure 5.7. The synthetic samples generated by the Generator network, colored as the corresponding groups.

5.3 Interpolation in GAN latent space

5.3.1 Interpolation using scRNA-seq data

Using the Jool dataset I performed interpolation between two cells: a real basal, and a real differentiated cell with 10 synthetic cells. I plotted four marker genes

if differentiation and showed how they change in pseudotime in Figure 5.8. I compared the results with the one obtained with the Monocle3 package in Figure 5.9, and found great correlations between them, except the *Itgb1* gene. The results show that the scsGAN model can produce similar results to other methods.

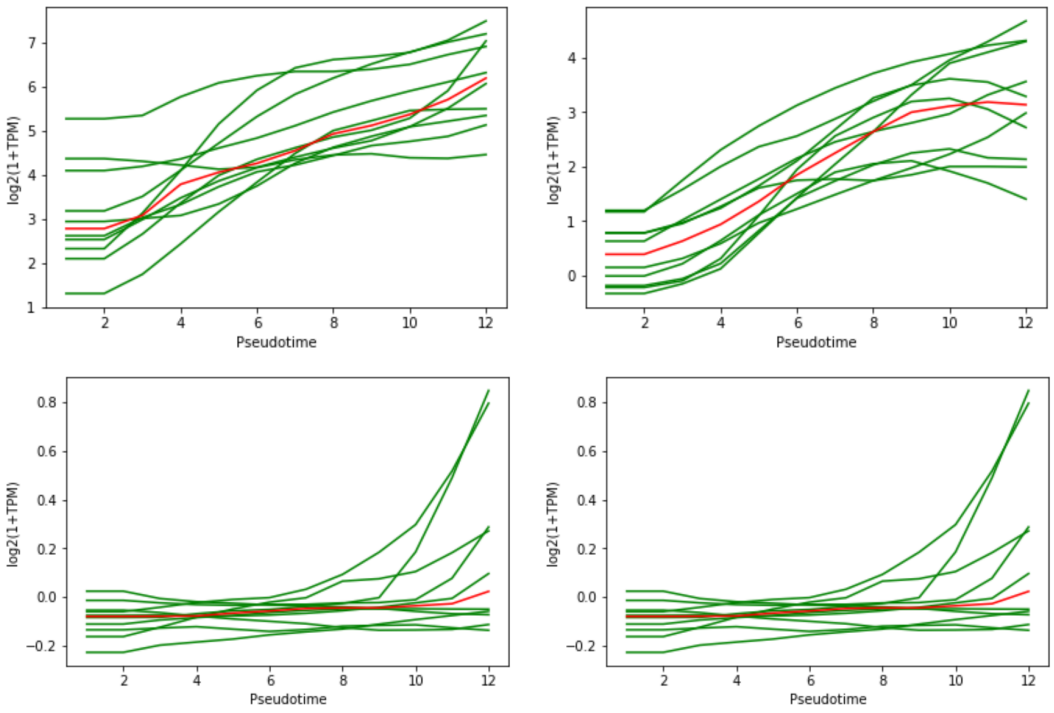


Figure 5.8. The changes of gene expression during differentiation in pseudotime, using the latent space interpolation. The green lines correspond to the single interpolated cell paths, and the red line is the median of the expressions in pseudotime. The figure shows the (from the top left to the bottom right) *Ppl*, *Grhl3*, *Itgb1*, *Uxs1* gene expressions.

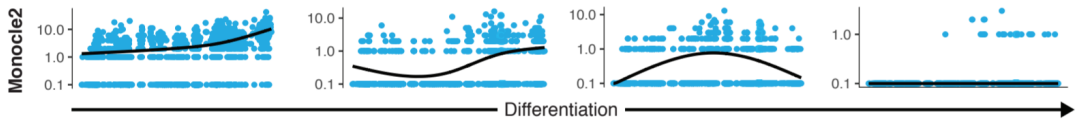


Figure 5.9. The changes in gene expression during differentiation in pseudotime, using the Monocle3 package. The blue dots are the real cells, and the black line is the mean expression. The figure shows the (from the left to right) *Ppl*, *Grhl3*, *Itgb1*, *Uxs1* gene expressions [44].

5.3.2 Interpolation using image-type data

The interpolation was performed on the images from the Galaxy Zoo dataset. Figure 5.10 shows how the interpolated galaxies look like between two differently shaped, and colored objects. The results are surprisingly good considering the networks in the GAN model were never designed for image processing (for example with convolutional layers), and also they are relatively small for a complex visual task like this. It can be seen, how both the color and the shape continuously change between the two real galaxies, producing realistic looking objects on the way.

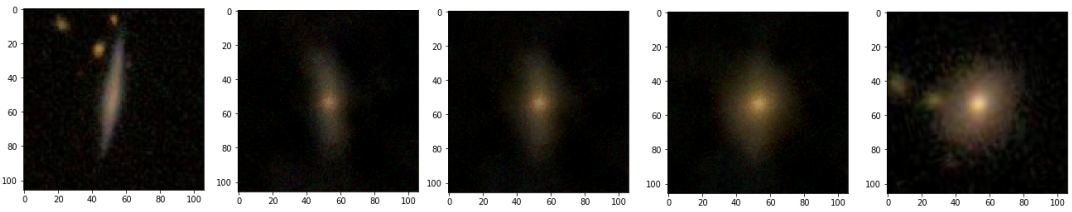


Figure 5.10. The results of the latent space interpolation between two real galaxies showed in image space.

5.4 Out of bag style transfer using latent space arithmetics

The style transfers were made from the unperturbated cells from the sixth group to the perturbated region of the sixth cells. The strength of the perturbation can be modified by setting the alpha parameter to different values. By setting the alpha parameter to 1 the model results in well executed style transfers as can be seen in Figure 5.11, where all of the cell groups and perturbation states are shown. In Figure 5.12 only the sixth group can be seen. The cells from the original data are colored red, and the unperturbated starting positions marked as green, and the perturbated ending positions as blue. The black arrows indicate the style transfer direction in UMAP space. The results with the alpha parameter set to 1 can be interpreted as nearly all of the starting cells landed on the perturbated region of the UMAP space. In Figure 5.13 a dendrogram shows the results of a hierarchical clustering, indicating that the perturbated synthetic cells did in fact transferred to the perturbated region in RNA expression space as well. The *alpha parameter* on the figure descriptions refer to the constant scalar value which the perturbation vector is multiplied by.

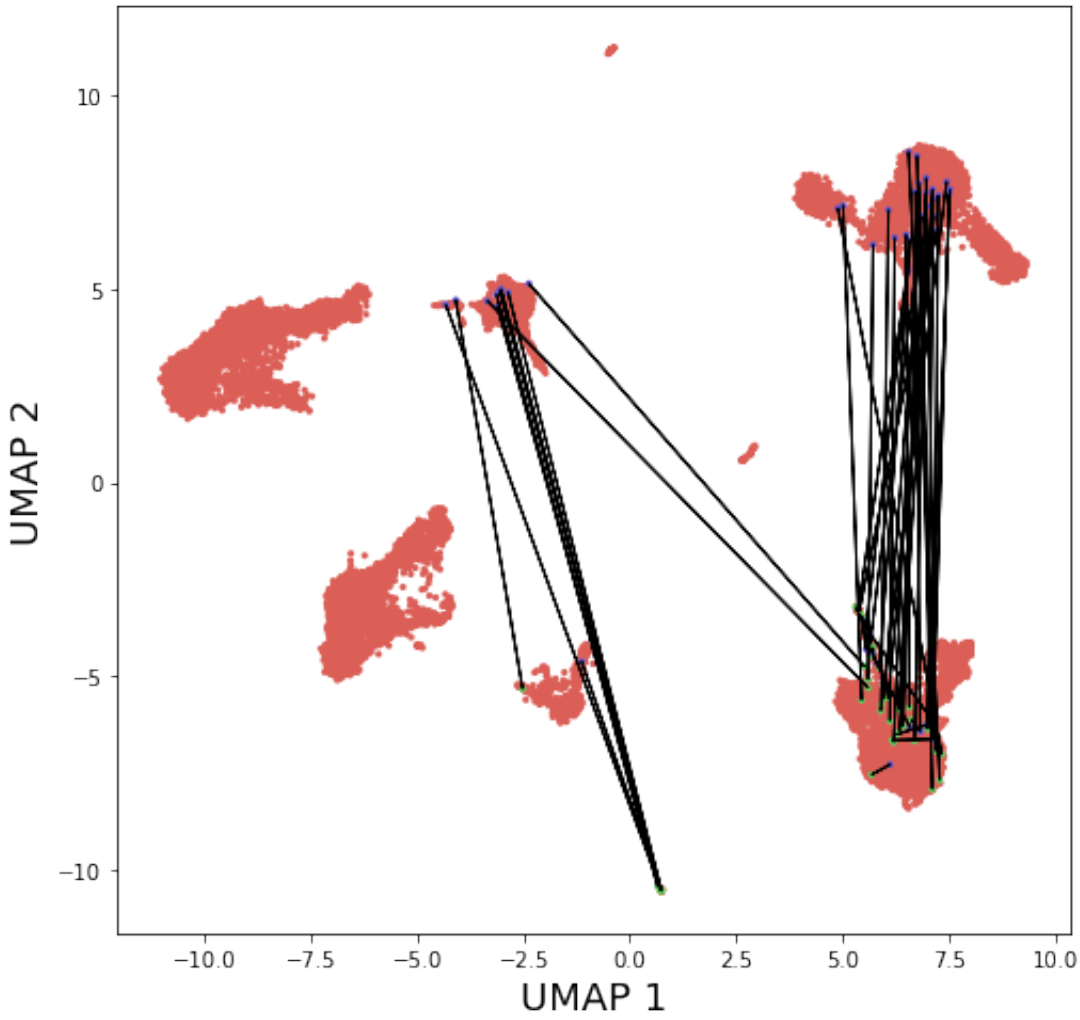


Figure 5.11. UMAP representation of the dataset with the corresponding perturbation vectors. Alpha parameter of the transformation was set to 1.

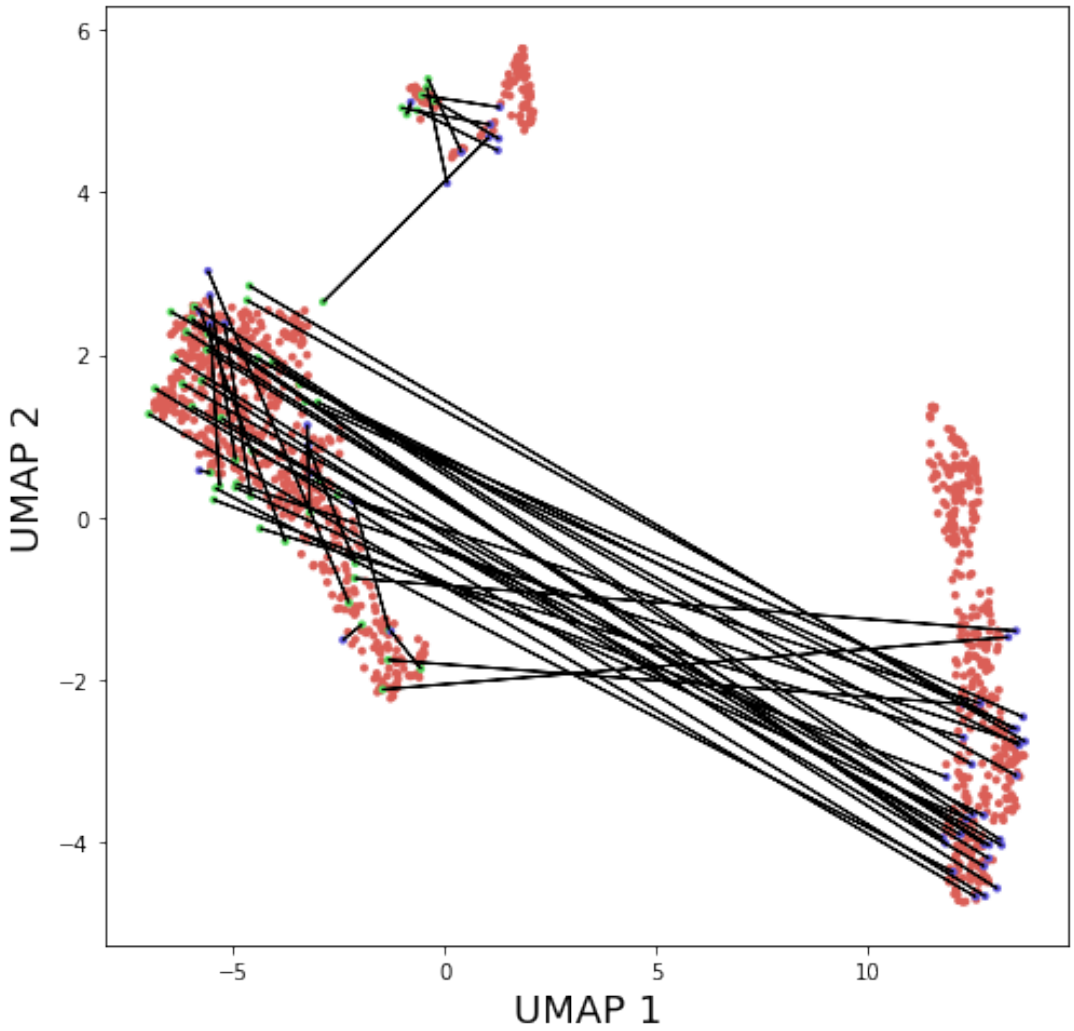


Figure 5.12. UMAP representation of the sixth group with the corresponding perturbation vectors. Alpha parameter of the transformation was set to 1.

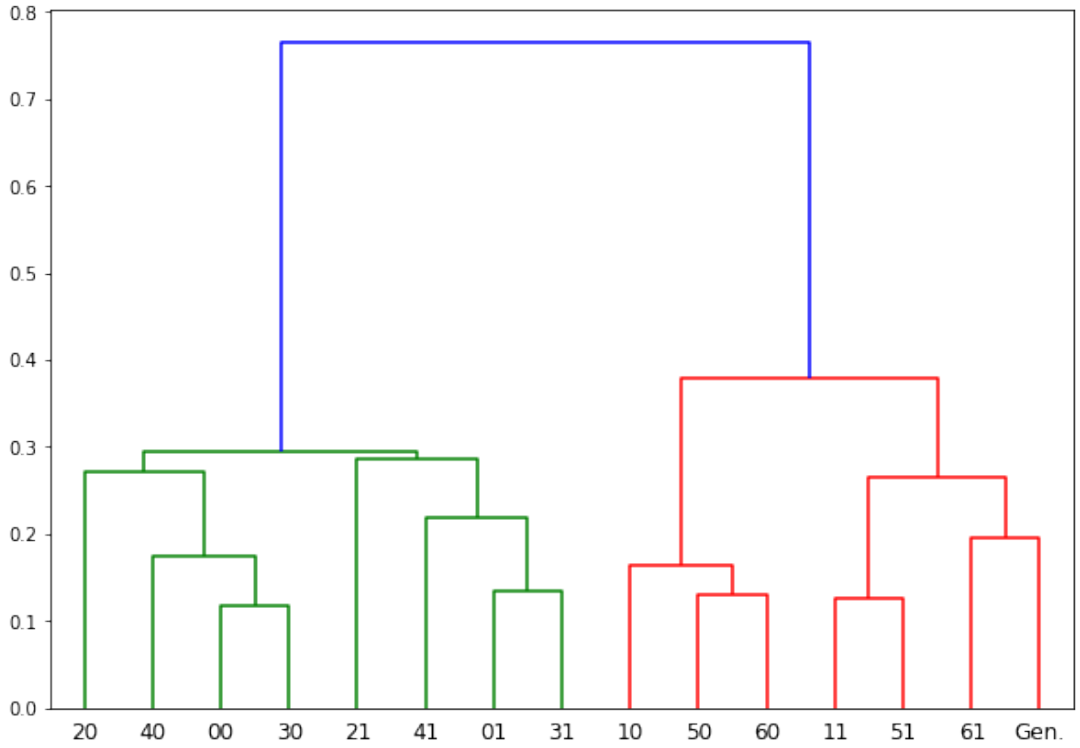


Figure 5.13. Dendrogram showing the results of the hierarchical clustering on the UMAP representation of the dataset with the corresponding perturbated cells labeled as *Gen.* Alpha parameter of the transformation was set to 1.

The following figures show the results of the same workflow, but with different alpha parameters. As one would expect, setting this parameter too low cause the style transfer to be failed, and setting it too high cause it to generate cells only in a very small part of both of the UMAP and gene expression spaces.

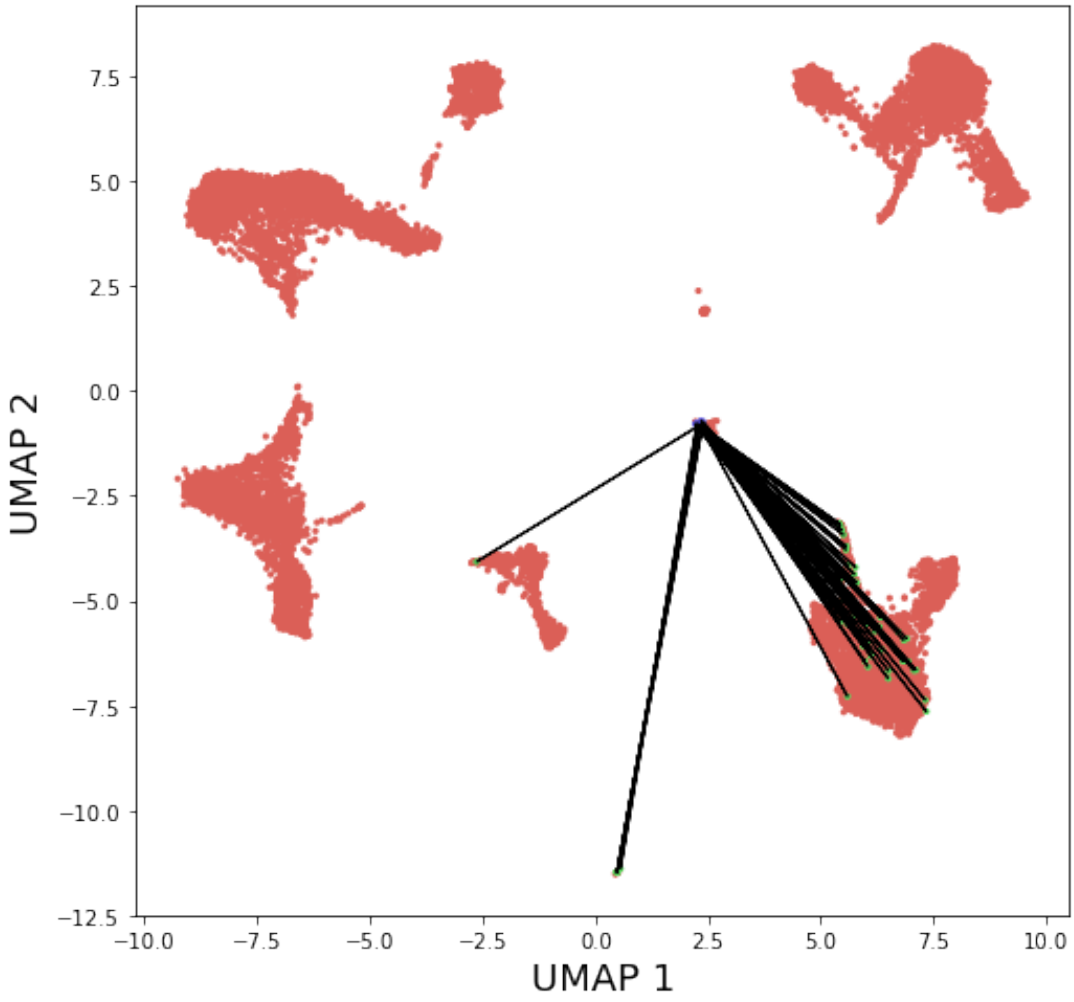


Figure 5.14. UMAP representation of the dataset with the corresponding perturbation vectors. Alpha parameter of the transformation was set to 50.

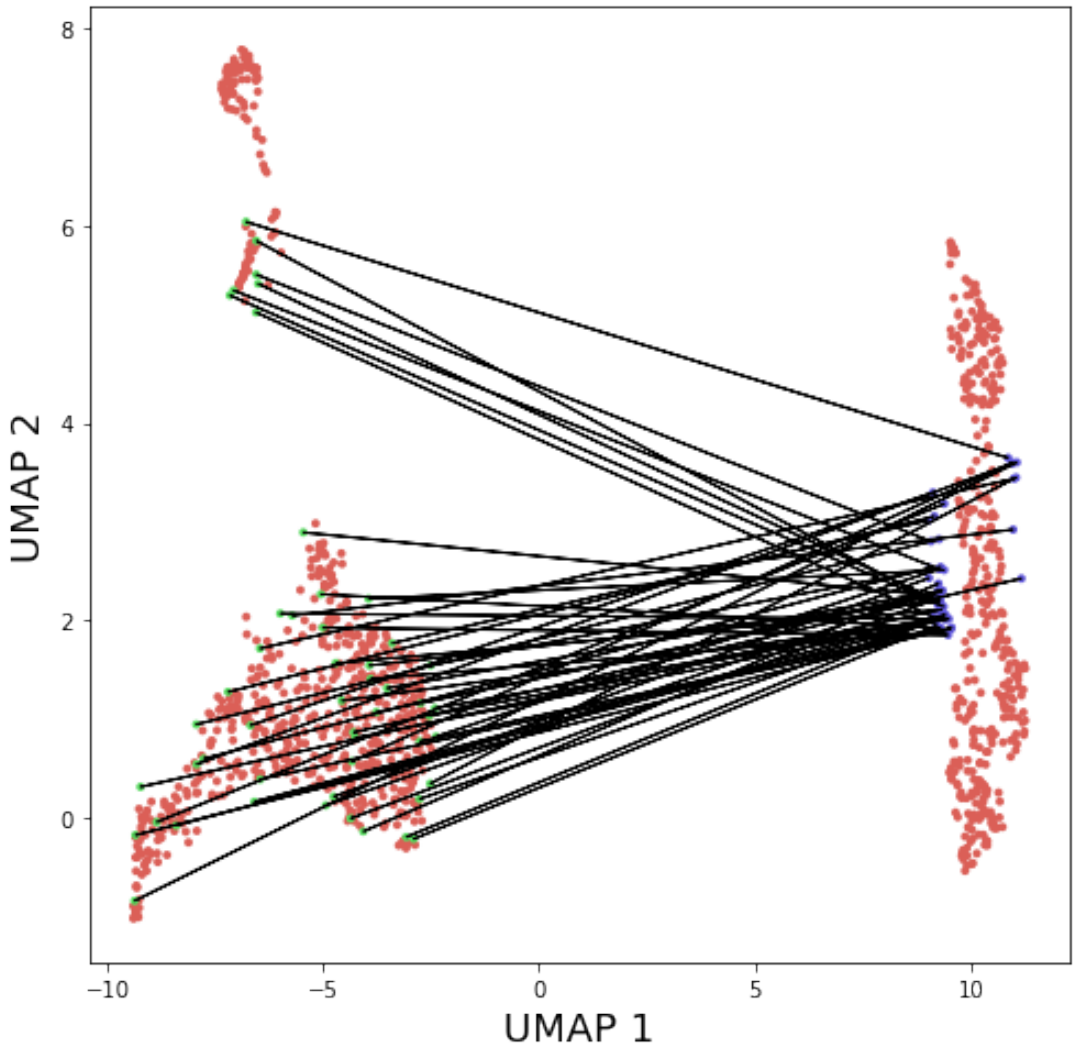


Figure 5.15. UMAP representation of the sixth group with the corresponding perturbation vectors. Alpha parameter of the transformation was set to 50.

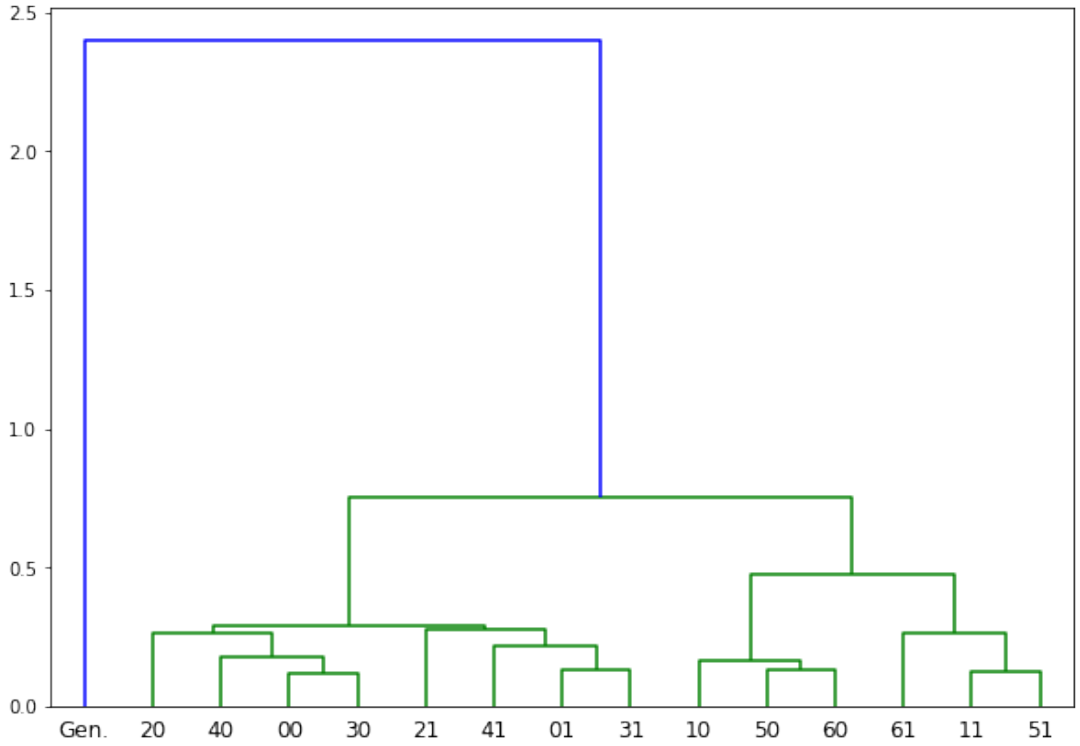


Figure 5.16. Dendrogram showing the results of the hierarchical clustering on the UMAP representation of the dataset with the corresponding perturbated cells labeled as *Gen.* Alpha parameter of the transformation was set to 50.

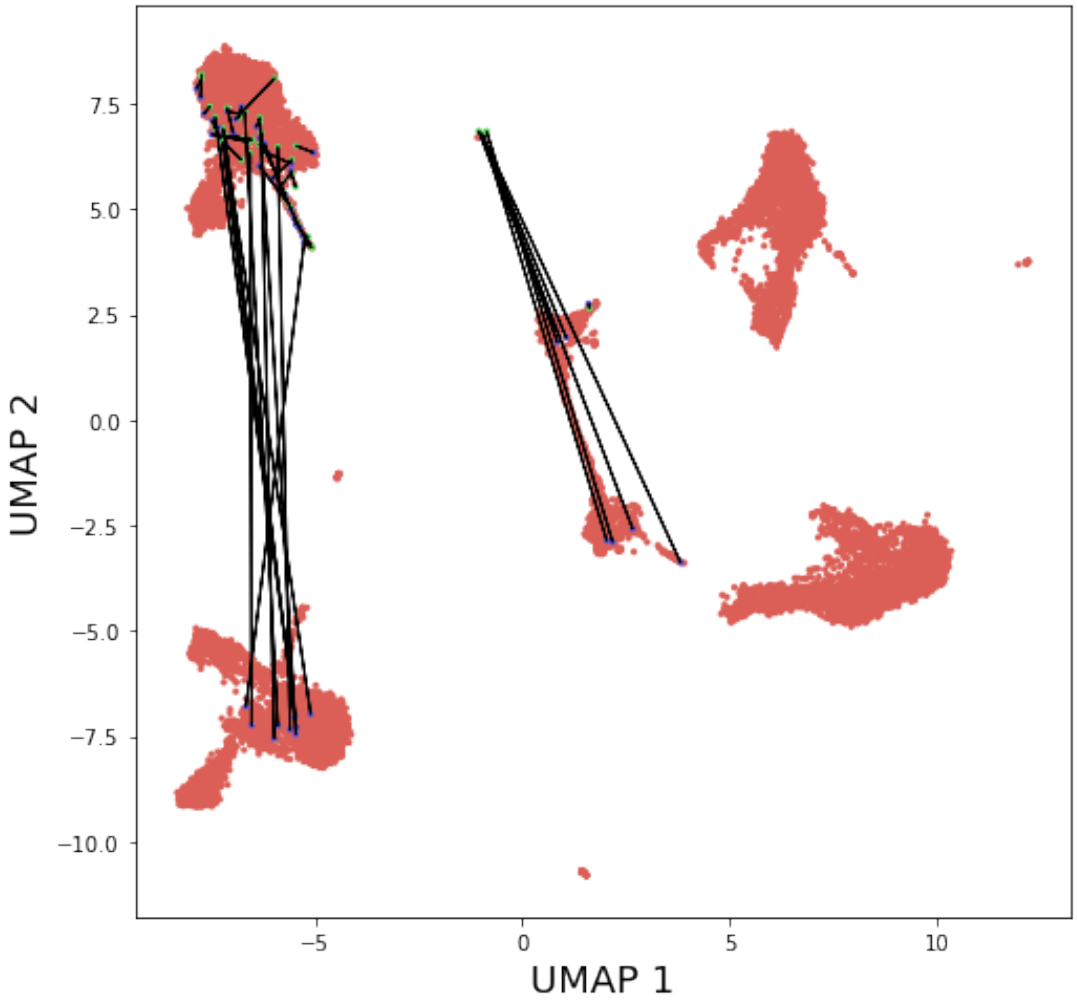


Figure 5.17. UMAP representation of the dataset with the corresponding perturbation vectors. Alpha parameter of the transformation was set to 0.5.

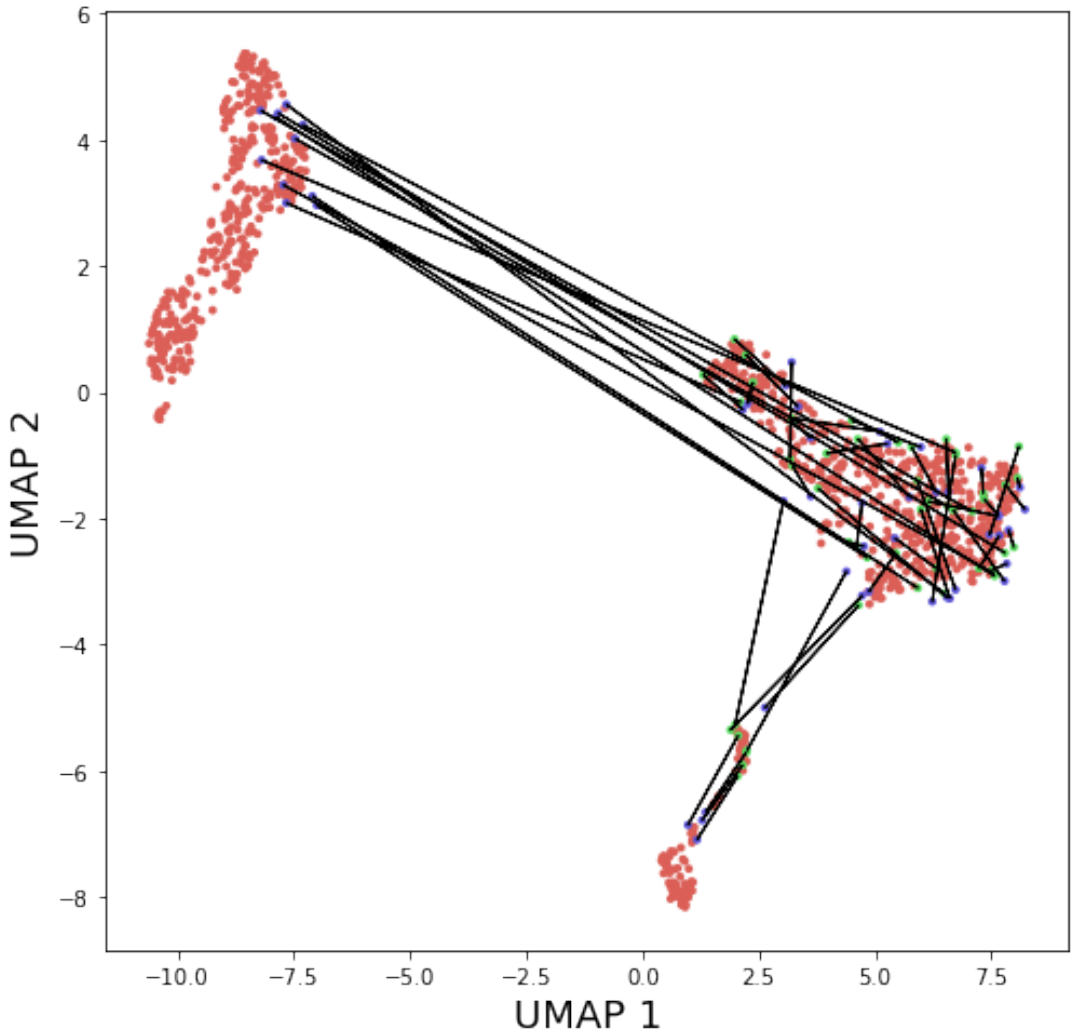


Figure 5.18. UMAP representation of the sixth group with the corresponding perturbation vectors. Alpha parameter of the transformation was set to 0.5.

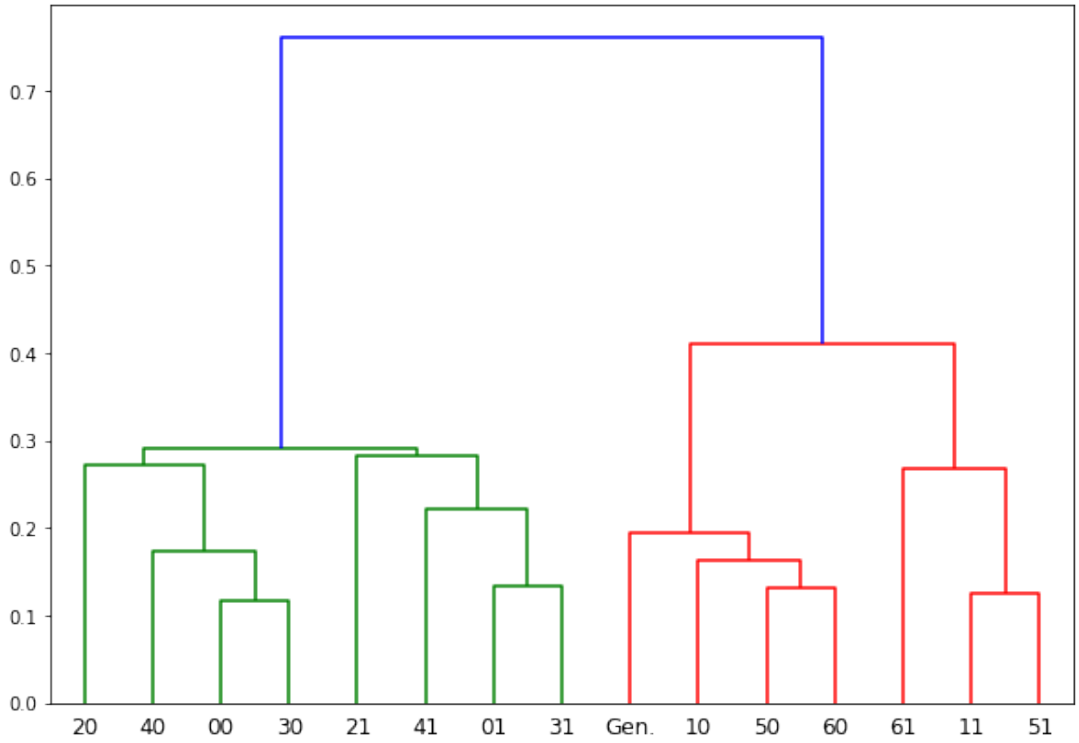


Figure 5.19. Dendrogram showing the results of the hierarchical clustering on the UMAP representation of the dataset with the corresponding perturbated cells labeled as *Gen.* Alpha parameter of the transformation was set to 0.5.

In Figure 5.20, Figure 5.20, Figure 5.20 can be seen what happens if one tries to execute style transfer with the alpha parameter set to 1, but the direction of the perturbation vector being randomized. The results show that it causes totally random perturbations, which indicates that the scsGAN model does, in fact, learns a meaningful perturbation vector.

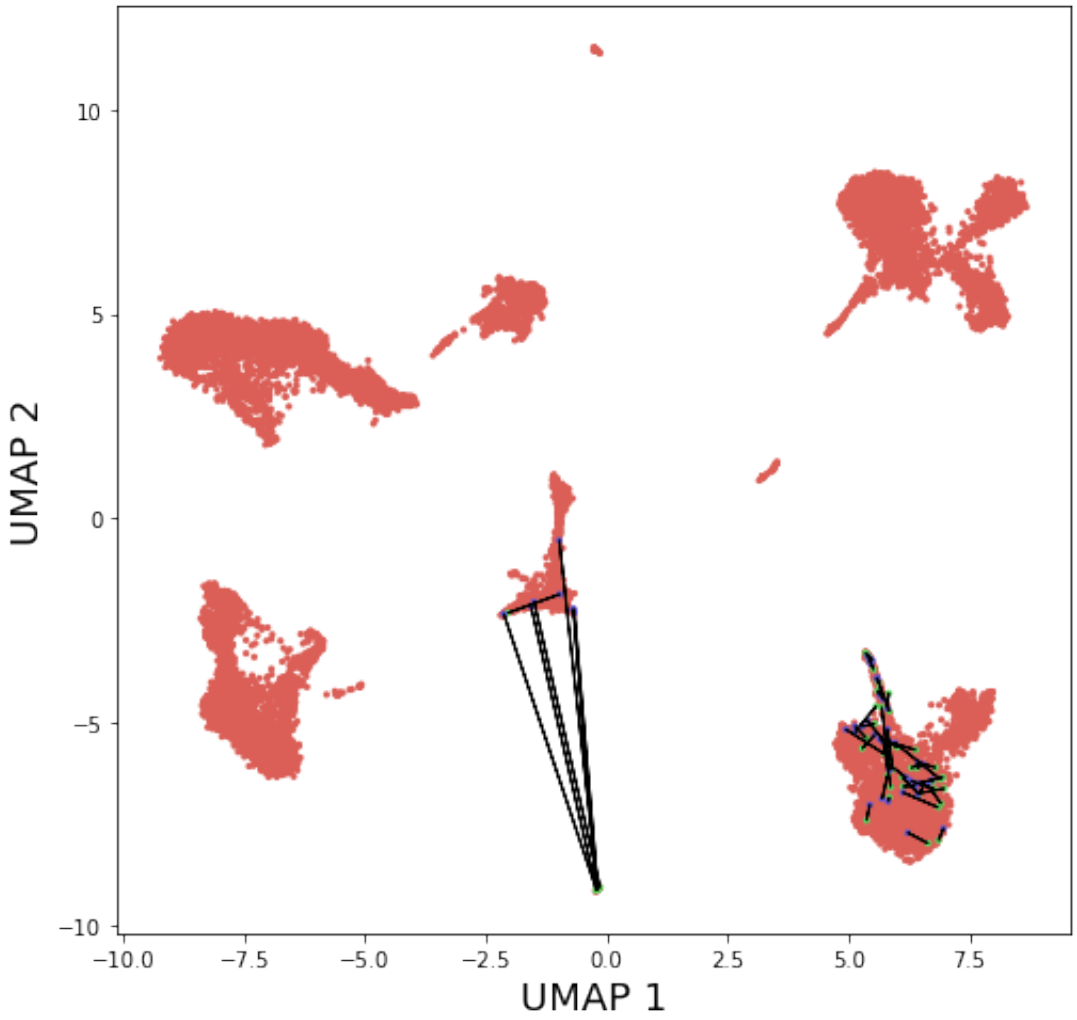


Figure 5.20. UMAP representation of the dataset with a random perturbation vectors. Alpha parameter of the transformation was set to 1.

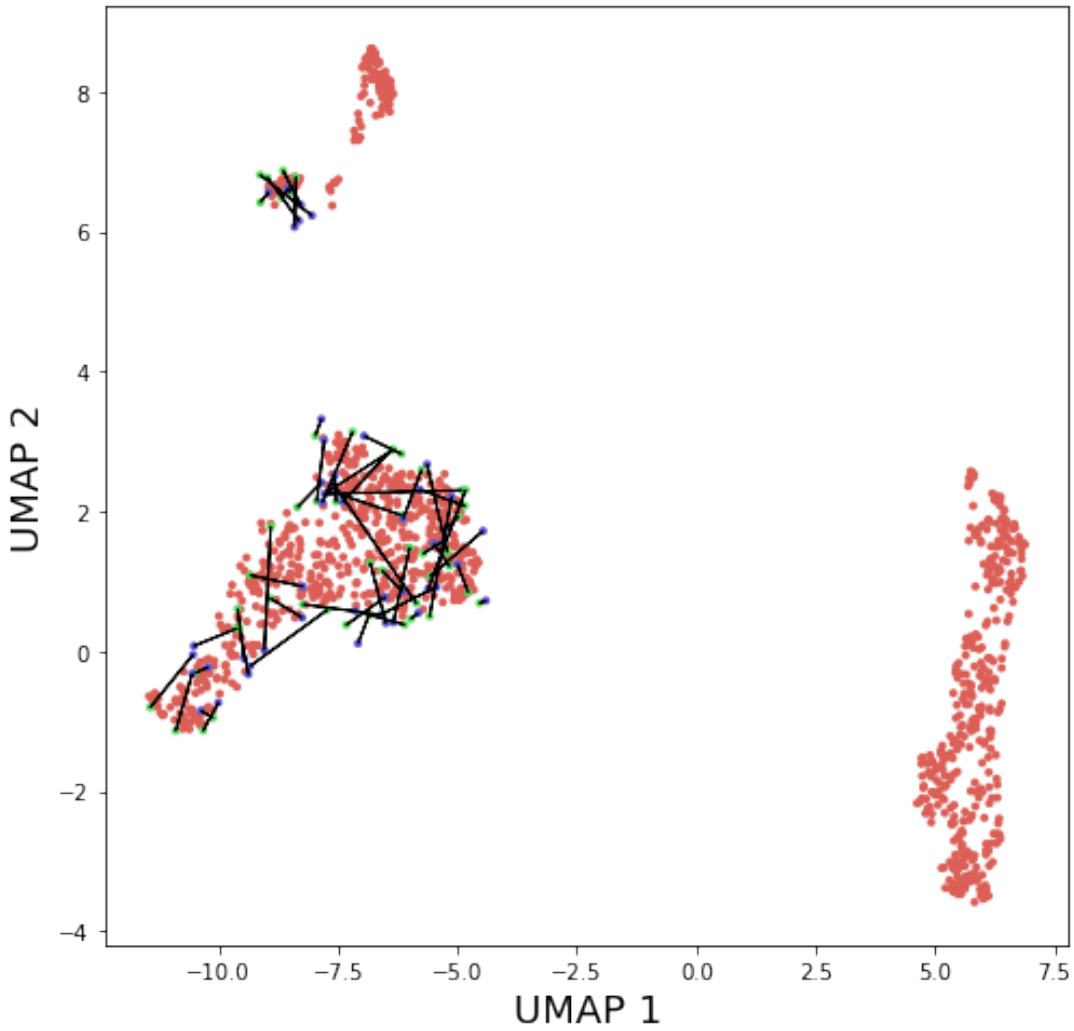


Figure 5.21. UMAP representation of the sixth group with a random perturbation vectors. Alpha parameter of the transformation was set to 1.

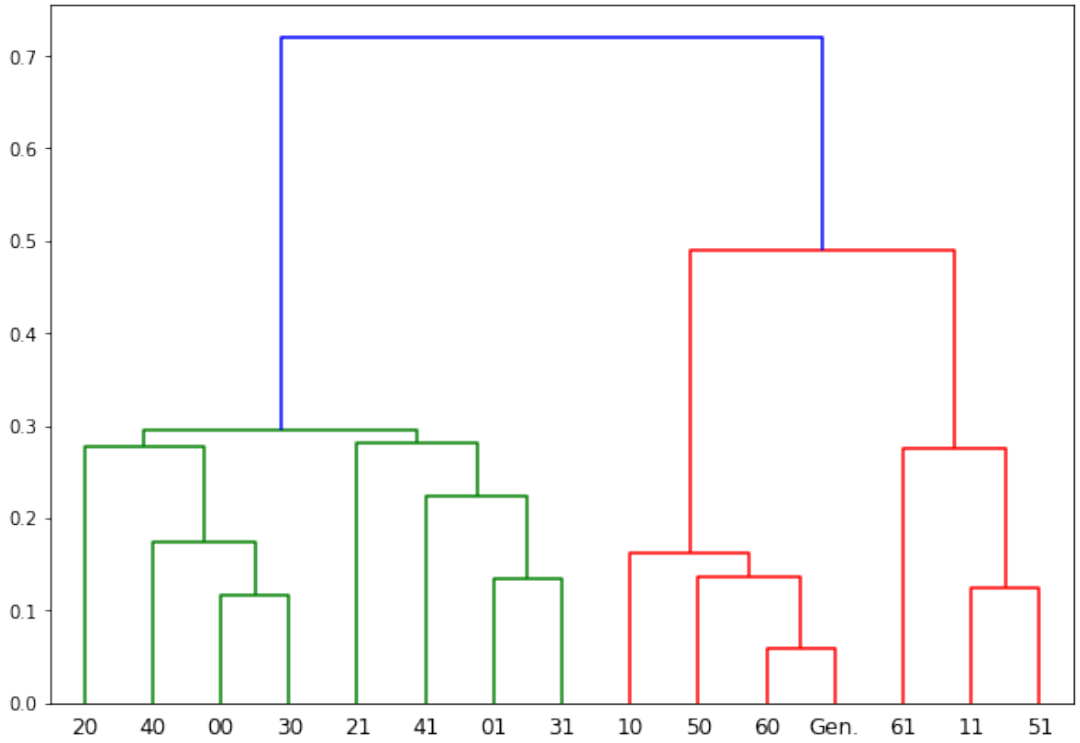


Figure 5.22. Dendrogram showing the results of the hierarchical clustering on the UMAP representation of the dataset a random corresponding perturbated cells labeled as *Gen*. Alpha parameter of the transformation was set to 1.

5.5 Dimensionality reduction using scsGAN

Using scsGAN for dimensionality reduction purposes resulted in a similar grouping of cells as with the PCA pre-processing step. However, the perturbation border, on the UMAP space is more isolated, there is less bridge-like structures between the two states as can be clearly seen in Figure 5.24 and Figure 5.26. This can be interpreted as the discriminator learns meaningful representations of the data.

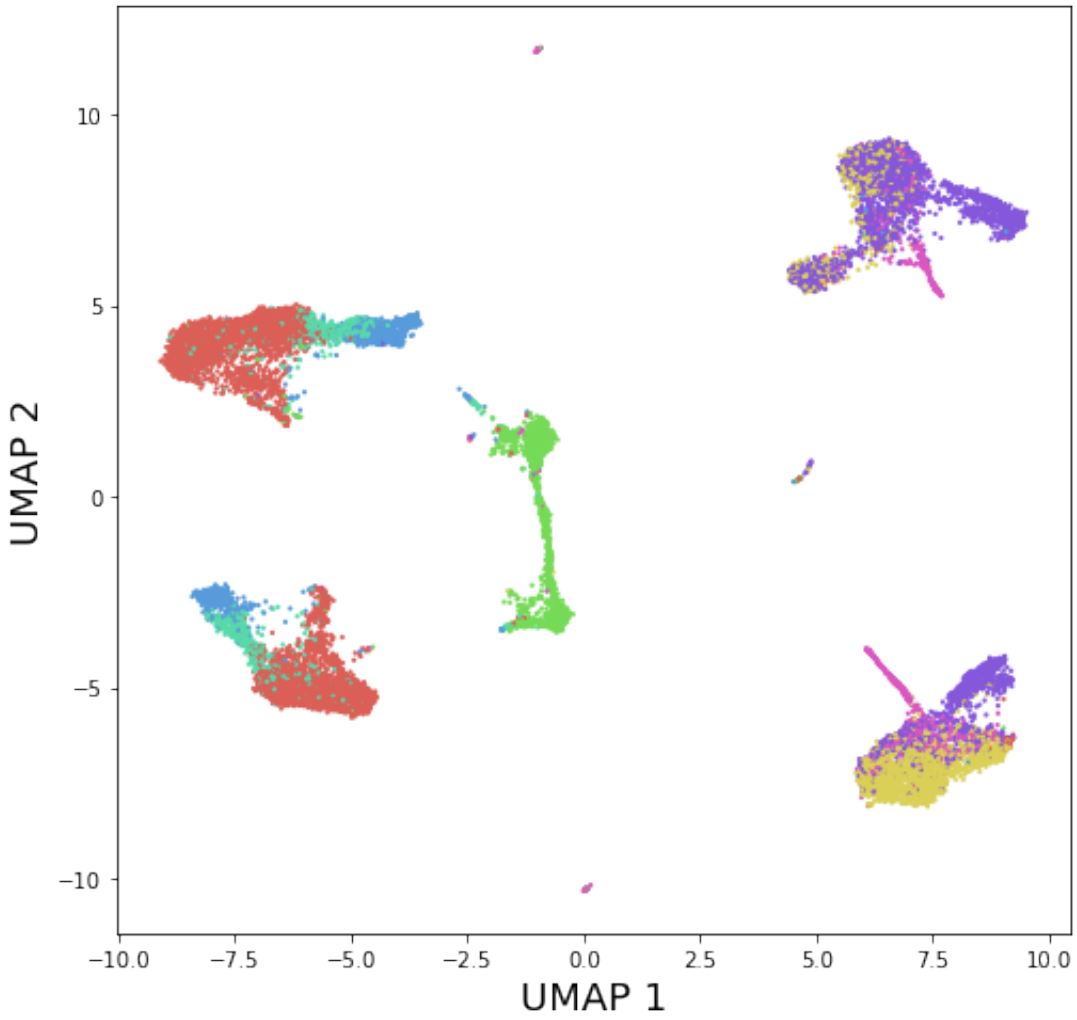


Figure 5.23. UMAP representation of the PBMC dataset, colored by the different group of cells. The pre-processing before the UMAP dimension reduction step was made by PCA.

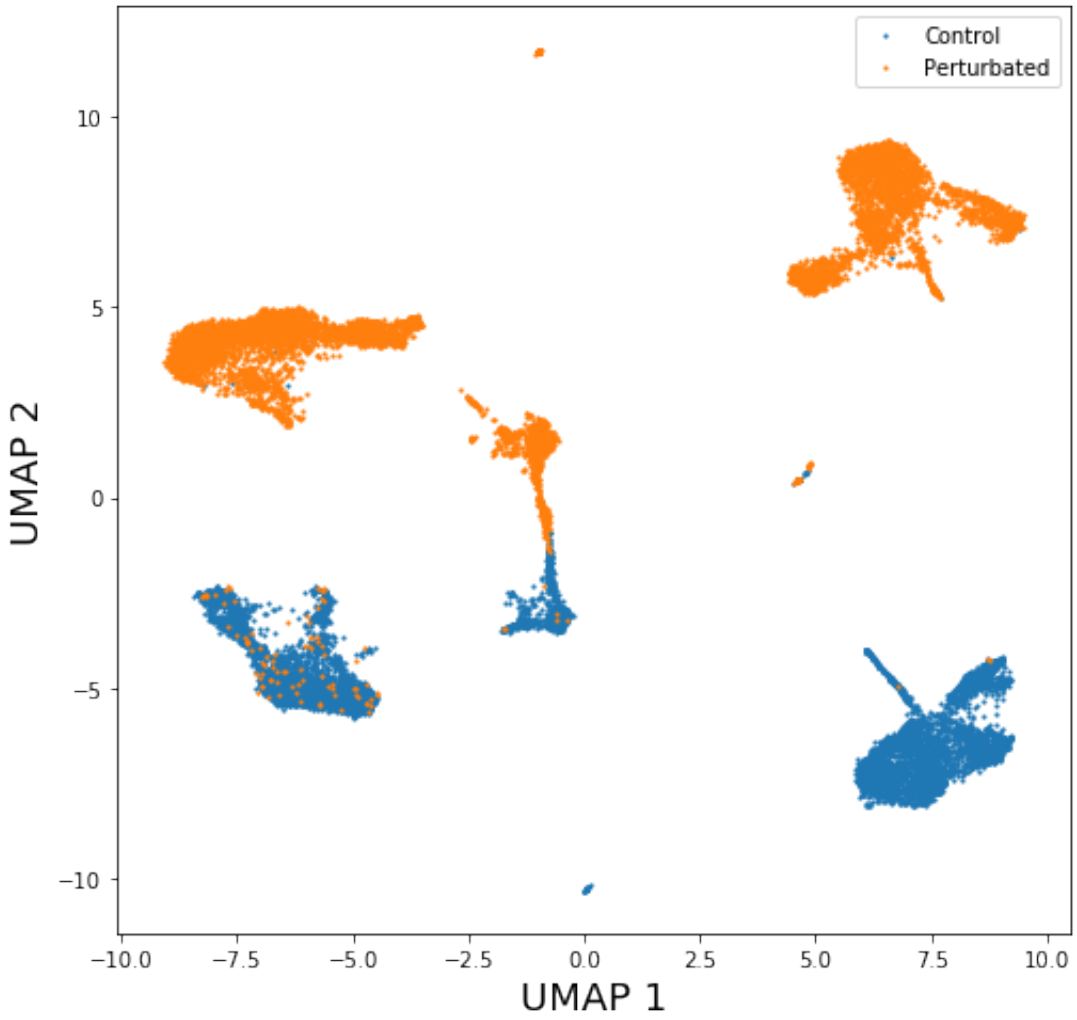


Figure 5.24. UMAP representation of the PBMC dataset, colored by the different type of treatment of the cells. The pre-processing before the UMAP dimension reduction step was made by PCA.

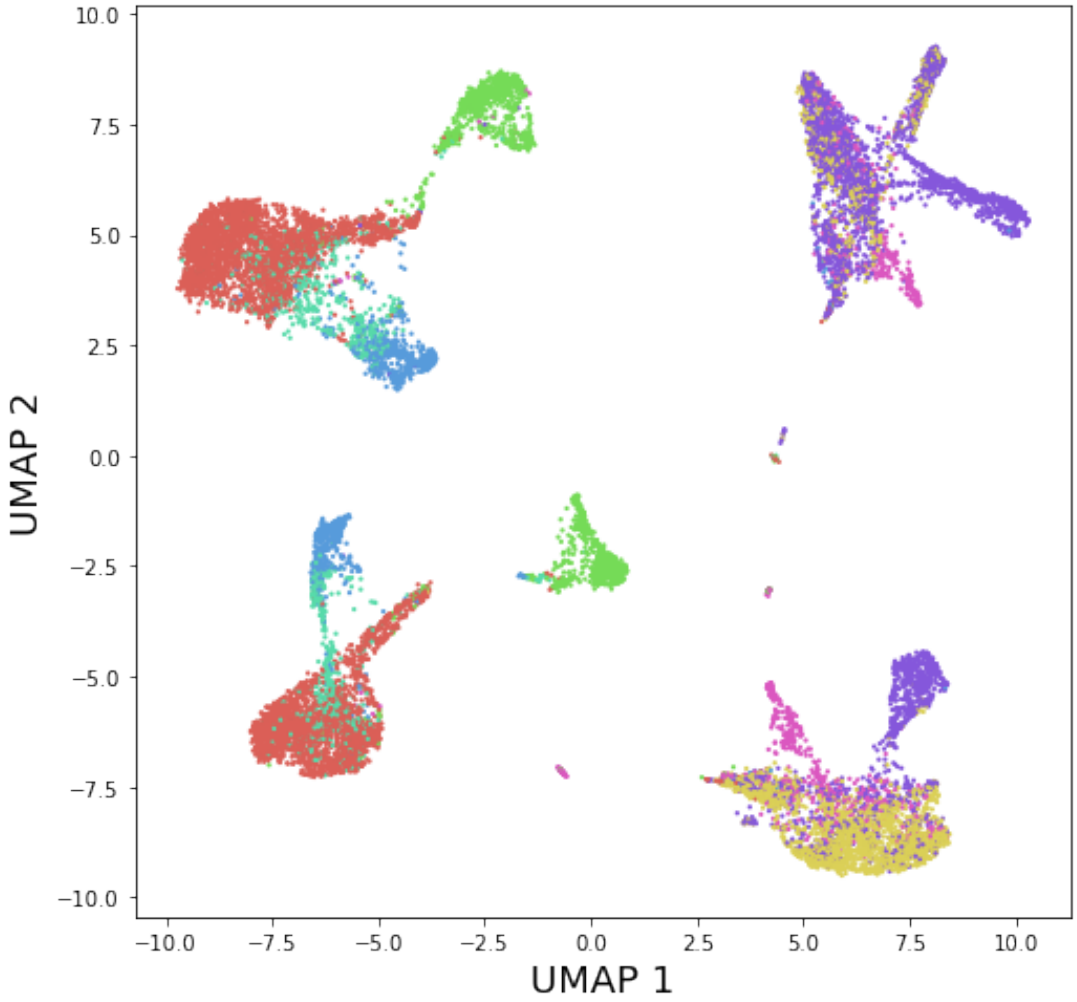


Figure 5.25. UMAP representation of the PBMC dataset, colored by the different group of cells. The pre-processing before the UMAP dimension reduction step was made by the GAN-Discriminator method.

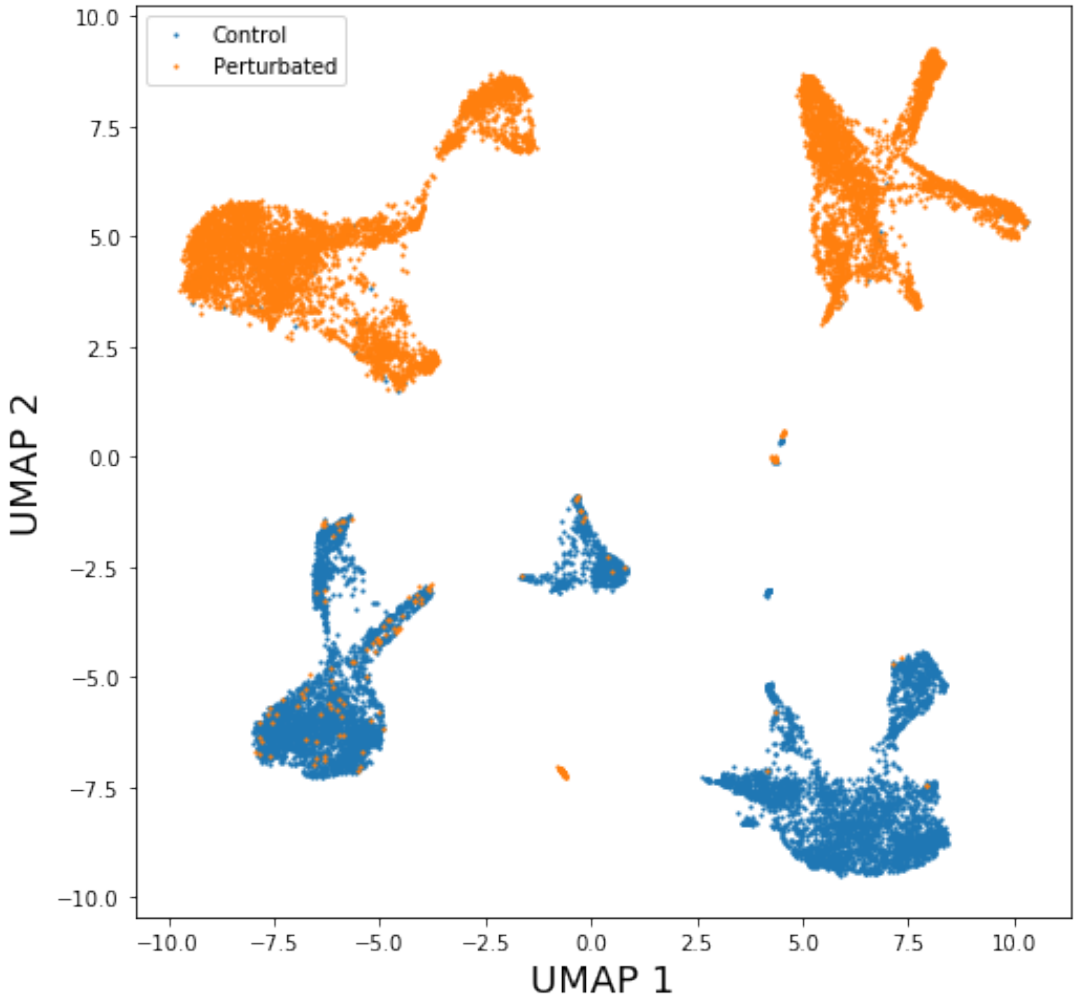


Figure 5.26. UMAP representation of the PBMC dataset, colored by the different type of treatment of the cells. The pre-processing before the UMAP dimension reduction step was made the GAN-Discriminator method.

Chapter 6

Conclusions

As the results in Chapter 5 suggest, scsGAN is capable to approximate scRNA-seq distributions even better than other regular methods used for the same purposes. It achieves this performance in a completely unsupervised and model-free way. It would be also interesting for future studies to explore the scalability of the method or to utilize the scsGAN in a semi-supervised manner.

Regarding the latent space inter- and extrapolation tasks, the model performs equivalently to regular methods, and accurately recreates the single-cell perturbations on both the gene expression and the UMAP reduced spaces. The fact that the model can generate realistic samples from other types of data structures, such as images shown in 5.3.2 without any specific optimization shows how well the model can be generalized.

The dimensionality reduction capabilities of the scsGAN showed comparable meaningful results to t-SNE and PCA methods, but needs to be further explored, as well as the data augmentation feature.

The developed workflow could open up new ways in the future to analyze single-cell data, in both retro and prospective ways. The next steps would include developing a user-friendly interface with the model and a visualization tool, which are both planned in the future.

References

[1] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019. 2

[2] S. Sengupta, S. Basak, P. Saikia, S. Paul, V. Tsalavoutis, F. Atiah, V. Ravi, and R. A. Peters, "A review of deep learning with special emphasis on architectures, applications and recent trends," *CoRR*, vol. abs/1905.13294, 2019. 2

[3] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt, and L. Zdeborova, "Machine learning and the physical sciences," 03 2019. 2

[4] A. Fischer and C. Igel, "An introduction to restricted boltzmann machines," pp. 14–36, 01 2012. 3

[5] M. Smith and J. Geach, "Generative deep fields: arbitrarily sized, random synthetic astronomical images through deep learning," 04 2019. 3

[6] D. Ribli, B. Pataki, and I. Csabai, "An improved cosmological parameter inference scheme motivated by deep learning," *Nature Astronomy*, vol. 3, 01 2019. 3

[7] D. Guest, K. Cranmer, and D. Whiteson, "Deep learning and its application to lhc physics," *Annual Review of Nuclear and Particle Science*, vol. 68, pp. 161–181, 10 2018. 3

[8] B. Hwang, J. H. Lee, and D. Bang, "Single-cell rna sequencing technologies and bioinformatics pipelines," *Experimental Molecular Medicine*, vol. 50, 08 2018. 3

[9] D. Hebenstreit, "Methods, challenges and potentials of single cell rna-seq," *Biology*, vol. 1, pp. 658–67, 12 2012. 3

[10] C. Targonski, B. T. Shealy, M. C. Smith, and F. A. Feltus, "Cellular state transformations using generative adversarial networks," 2019. 4

[11] S. Rashid, S. Shah, Z. Bar-Joseph, and R. Pandya, "Dhaka: variational autoencoder for unmasking tumor heterogeneity from single cell genomic data," *Bioinformatics*, 02 2019. btz095. 4

[12] H. Wang, B. Raj, and E. P. Xing, "On the origin of deep learning," *CoRR*, vol. abs/1702.07800, 2017. 7

[13] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016. 7, 12

[14] J. B. Ahire, "The artificial neural networks handbook: Part 4," *Medium*, Nov 2018. 8

[15] E. Charniak, "Introduction to deep learning," 2019. 9

[16] M. Bursac, D. Milosevic, and K. Mitrović, "Proposed model for automatic learning style detecting based on artificial intelligence," 05 2019. 10

[17] H. Li, Z. Xu, G. Taylor, and T. Goldstein, "Visualizing the loss landscape of neural nets," 2017. 13

[18] I. Dabbura, "Gradient descent algorithm and its variants," 14

[19] Y. Lecun, "A theoretical framework for back-propagation," 08 2001. 14

[20] D. Stansbury, "Derivation: Error backpropagation gradient descent for neural networks," 15

[21] A. Kathuria, "https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/," *Paperspace Blog*, Aug 2018. 17

[22] A. Graves, "Generating sequences with recurrent neural networks," *Neural and Evolutionary Computing*, 08 2013. 17

[23] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010. 17

[24] C. Yinka-Banjo and O.-A. Ugot, "A review of generative adversarial networks and its application in cybersecurity," *Artificial Intelligence Review*, 06 2019. 19, 20, 23

[25] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv:1411.1784*, 11 2014. 20, 26

[26] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," 2014. 21

[27] "Illustration of the gan framework," *NVIDIA Developer Blog*. 22

[28] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *ArXiv*, vol. abs/1701.07875, 2017. 24

[29] P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam, "Optimizing the latent space of generative networks," 07 2017. 27, 28

[30] A. AlJanahi, M. Danielsen, and C. Dunbar, "An introduction to the analysis of single-cell rna-sequencing data," *Molecular Therapy - Methods and Clinical Development*, vol. 10, pp. 189–196, 09 2018. 29, 30

[31] A. Peixoto, M. Monteiro, B. Rocha, and H. Veiga-Fernandes, "Quantification of multiple gene expression in individual cells," *Genome research*, vol. 14, pp. 1938–47, 11 2004. 29

[32] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I. Kwok, L. G. Ng, F. Ginhoux, and E. Newell, "Dimensionality reduction for visualizing single-cell data using umap," *Nature Biotechnology*, vol. 37, 12 2018. 31

[33] "How umap works," *How UMAP Works - umap 0.3 documentation.* 32, 33, 34, 35, 36, 37, 38, 39

[34] L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," *J. Open Source Software*, vol. 3, p. 861, 2018. 35, 41

[35] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," pp. 577–586, 2011. 40

[36] L. Zappia, B. Phipson, and A. Oshlack, "Splatter: Simulation of single-cell rna sequencing data," *Genome Biology*, vol. 18, 12 2017. 41, 44, 51

[37] X. Qiu, Q. Mao, Y. Tang, L. Wang, R. Chawla, H. Pliner, and C. Trapnell, "Reversed graph embedding resolves complex single-cell trajectories," *Nature methods*, vol. 14, 08 2017. 45

[38] P.-Y. Tung, J. Blischak, J. Hsiao, D. Knowles, J. Burnett, J. Pritchard, and Y. Gilad, "Batch effects and the effective design of single-cell gene expression studies," *Scientific Reports*, vol. 7, 01 2017. 50

[39] I. Engel, G. Seumois, L. Chavez, D. Samaniego-Castruita, B. White, A. Chawla, D. Mock, and P. Vijayanand, "Innate-like functions of natural killer t cell subsets result from highly divergent gene programs," *Nature Immunology*, vol. 17, 04 2016. 50

[40] M. Lotfollahi, F. Wolf, and F. Theis, "scgen predicts single-cell perturbation responses," *Nature Methods*, vol. 16, pp. 715–721, 08 2019. 50, 52, 53, 54

[41] S. Joost, A. Zeisel, T. Jacob, X. Sun, G. La Manno, P. Lönnerberg, S. Linnarsson, and M. Kasper, "Single-cell transcriptomics reveals that differentiation and spatial signatures shape epidermal and hair follicle heterogeneity," *Cell Systems*, vol. 3, pp. 1–17, 09 2016. 50

[42] "https://zoo4.galaxyzoo.org/," 50

[43] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, p. 60, Jul 2019. 54

[44] A. Ghahramani, F. Watt, and N. Luscombe, "Generative adversarial networks simulate gene expression and predict perturbations in single cells," 2018. 66