TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALEKSI HIETANEN

A SCALABLE METHOD FOR NON-LINEAR DIMENSIONALITY REDUCTION WITH APPLICATIONS TO SINGLE-CELL DATA

Master's thesis

Examiner: John Doe

The examiner and topic of the thesis were approved on 11 September 2017

ABSTRACT

ALEKSI HIETANEN: A Scalable Method for Non-Linear Dimensionality Reduction with Applications to Single-Cell Data
Tampere University of Technology
Master of Science Thesis, 54 pages, 2 Appendix pages
October 2018
Master's Degree Programme in Science and Engineering
Major: Theoretical computer science
Examiner: John Doe

Keywords: Key, Word, List

TODO

TIIVISTELMÄ

ALEKSI HIETANEN: Otsikoimaton
Tampereen teknillinen yliopisto
Diplomityö, 54 sivua, 2 liitesivua
Lokakuu 2018
Teknis-luonnontieteellinen diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotiede
Tarkastaja:

Avainsanat:

TODO

PREFACE

TODO

In Helsinki, Finland, on December 12, 2018

Aleksi Hietanen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AE	autoencoder
PCA	principal component analysis
GPU	graphics processing unit
GPGPU	general-purpose computing on graphics processing units
TPU	tensor processing unit
MSE	mean squared error
SGD	stochastic gradient descent
VI	variational inference
MCMC	Markov chain Monte Carlo
RBM	restricted Boltzmann machine
VAE	variational autoencoder
GAN	generative adversarial network
UMAP	uniform manifold approximation and projection
SNE	stochastic neighbor embedding
ReLU	rectifier linear unit
t-SNE	t-distributed stochastic neighbor embedding
k-NN	k-nearest neighbors
RNA	ribonucleic acid
scRNA-seq	single-cell RNA sequencing

$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$p(x\|y)$	conditional probability of x given y
$\sim$	is distributed as
$\frac{\partial}{\partial x}$	partial derivative w.r.t. x
∇f	gradient of f
$H(p)$	entropy of a random variable with distribution p
$H(p, q)$	cross-entropy of distributions p and q
$D_{\mathrm{KL}}(p \parallel q)$	Kullback-Leibler divergence from q to p
$\mathbb{E}_p[q]$	Expectation of q under distribution p
$\mathcal{O}(f)$	asymptotic upper bound of f

1. INTRODUCTION

[Note: Most of this chapter is probably going to be rewritten]

At present, ever larger sets of increasingly higher dimensional data are being produced, gathered and analyzed. As a consequence, several challenges arise when analyses are carried out on these vast amounts of high dimensional data. This recent surge of data prompts not only for the development of methods for data analysis capable of scaling to massive data sets but ones which are simultaneously capable of dealing with challenges inherent to high dimensional data.

In literature, problems arising from high dimensionality are collectively referred to as the *curse of dimensionality*. Particularly in machine learning the following aspects of high dimensionality are problematic

- distances between pairs of points become less distinguished as dimensionality of the space increases [2], rendering methods based on distance metrics at a disadvantage.
- The volume of a space grows exponentially along with the number of dimensions. This leads to even large data sets appearing sparse in the feature space. Efficient sampling requires a lot of points
- need for more training data, k-means example [75, p. 263]
- algorithm complexity depends on the dimensionality of data being processed
- from a more practical perspective, high dimensional data by itself is hard to directly reason about and difficult to visualize in any meaningful sense, as simple statistics will often not be able to capture adequate information of the data distribution.

The given listing is not exhaustive, but rather serves as an illustration for the variety of issues that can be encountered when working with high dimensional data.

To tackle problems associated with high dimensional data a common approach is to seek for a lower dimensional representation of the data which retains its salient features. This process, known as dimensionality reduction, can be split into two different approaches, namely *feature selection* and *feature extraction*.

The so-called manifold hypothesis [62] often holds for real-world data sets. The manifold hypothesis posits that the high dimensional data in question lies near a much lower dimensional manifold. As an example, take ... . This in turn implies that if it is possible to learn this manifold, the original data could be described in far fewer dimensions with minimal loss of information. This, together with the classical Johnson-Lindenstrauss

lemma [38], which provides general bounds to the error associated with dimensionality reduction between points of two Euclidean spaces, give a strong indication that for many data sets substantial dimensionality reduction is not only feasible, but that most of the information within the data can be captured with fewer parameters.

Several different methods for dimensionality reduction have been developed over the years. Arguably the most well known and applied dimensionality reduction method in practice is Principal Component Analysis (PCA), developed independently by Karl Pearson [20] and Harold Hotelling [34] in the early 1900s. Although very efficient to compute and providing easily interpretable results, PCA however suffers from the fact that it only takes into account linear relationships in the data it is given, making it nonideal for ... . The fact that PCA only aims to find an orthogonal projection that maximizes the variance... local structure of the data is poorly preserved. On the other hand, methods that aim to preserve local structure of data typically suffer from poor scalability to large data sets and the curse of dimensionality in several ways, such as ... .

1.1 Scope and Objective

As a wealth of dimensionality reduction methods have been developed, attempting to cover all of them would be greatly out of the scope of this thesis, in which the focus is on improving an existing method for primarily data visualization purposes. For comprehensive reviews on other prominent methods the reader is referred to [56, 7].

In exploratory data analysis, visualization of the characteristics of data sets plays a key role. Being able to project high dimensional data into meaningful low dimensional representations enables researchers and data analysts to generate hypotheses.

In this work, a method for improving the applicability of parametric t-SNE is proposed. In the proposed method a generative model in the form of a Variational Autoencoder is first trained and subsequently used to produce training data for the parametric t-SNE embedding. The benefits of this procedure are severalfold: . Further, each of these benefits are experimentally demonstrated on a variety of data sets, both quantitatively and qualitatively.

1.2 Outline

The structure of this thesis is as follows. In Chapter 2 the relevant background and related work to the contribution of this thesis is covered to give context to an audience not previously familiar with variational autoencoders or the t-SNE method for dimensionality reduction. The chapter is organized into two main sections, one that builds up to the theory and recent advances in variational autoencoders from the introduction of neural networks, and the other section presents the development of t-SNE up to its parametrization with neural networks.

The method developed for this thesis is presented in Chapter 3. The proposed learning and inference algorithms are presented, together with discussion on extensions and further variations. In addition, relevant implementation details of the application built to produce the results for the experiments chapter are briefly covered.

In Chapter 4 the proposed method is tested experimentally for different properties presented in the previous chapter, as well as compared with existing methods. The experiments chapter includes descriptions for the various data sets being used and definitions for the metrics employed to quantitatively assess the quality of the produced low dimensional embeddings by our method and other methods under comparison.

The final chapter presents discussion and conclusions for the work carried out in this thesis. Additionally, directions for future work are suggested.

2. BACKGROUND AND RELATED WORK

[Note: Some general background in more detail than the intro? Ideas listed below]

- main motivator for this work is to provide a tool to enhance exploratory analysis in cancer research
- on molecular data
- add references where visualization is used to aid analysis
- discuss current state of the art
- challenges associated with currently used methods
- we outline in this chapter the relevant background information on the two methods that are used in the developed method, discussed in chapter 3

2.1 Variational Autoencoders

An essential component of the method presented in this work are Variational Autoencoders (VAE). VAEs belong to a broad class of models termed deep generative models, which have in recent years played an important role in machine learning research. As their name suggests, generative models attempt to model the process by which a given set of data has been generated by learning the underlying distribution of the data. Deep generative models are distinguished by the property that they are parameterized by neural networks, enabling the use of deep learning methods.

By learning the underlying distribution of data sets it is possible to perform a variety of inference tasks, such as missing data imputation, classification and . In addition, having insight into the data generating process, one can generate new data points of high similarity.

[Note: short history of methods developed, RBM/DBN/DBM/GAN...] [19, 73, 30, 29, 25].

[Note: Thorough review of deep generative models mentioned here, but which are out of scope for this work, can be found in [24, Chapter 20].]

Before moving on to specialized artificial neural network structures and learning objectives, a brief review of the relevant background literature is presented. First, a general introduction to artificial neural networks is given, after which autoencoder neural networks are discussed in more detail. The theory and purpose of variational Bayesian methods are discussed to finally arrive at the full definition of VAEs by bringing together the contents of the preceding sections.

2.1.1 Artificial Neural Networks

Artificial neural networks, or as commonly referred to simply as neural networks, are a computational framework inspired by the biology of animal brains, more specifically the interconnected structure of individual nerve cells. Not unlike their biological counterparts, neural networks are composed of individual neurons capable of processing and passing along signals from one another to perform complex computation tasks as a whole.

Neural networks are at the core of many recent advances in machine learning. Particularly, deep neural network architectures now hold many state-of-the-art results in fields including computer vision, natural language processing, and speech recognition [49]. More broadly, neural networks have been applied to all different categories of machine learning tasks, i.e. supervised, semi-supervised and unsupervised learning, with great success.

Advances in supervised learning for computer vision have arguably played an important role in the development of modern machine learning. A turning point for deep learning was the success of a deep convolution neural network architecture [47], later dubbed AlexNet, which performed considerably better than its competition in the 2012 ImageNet Large Scale Visual Recognition Challenge [15, 72].

On the unsupervised learning front, deep belief networks, or rather unsupervised pretraining by stacking RBMs is one of the most important results in reviving interest in deep learning, as it was one of the first generally applicable results to enable learning of deep networks by means of better initialization of network parameters. More generally, learning meaningful representations from unlabeled data is an active research area, where methods of deep learning are currently producing state-of-the-art results with models such as the VAE, GAN and their variants.

Recently, deep learning in conjunction with reinforcement learning has produced superhuman results on tasks where previous methods of artificial intelligence have failed, such as playing video games [61], the game of go [76] and shortly after, a fully unsupervised variant capable of generalizing to other games, convincingly beating previously developed chess and shogi engines [78, 77].

The development of ever faster hardware for accelerating the computations needed for deep learning has played a pivotal role in the resurgence of research interest in neural networks. Originally built to accelerate the linear algebra operations required to produce 3D computer graphics, graphics programming units (GPU) offer immense parallel computing capabilities compared to traditional central processing units (CPU). With the advent of programming languages and APIs targeting GPUs, general-purpose computing on this specialized hardware has been made possible and has subsequently been taken advantage of in research and development of applications making use of neural networks. Recently, hardware specialized solely for use with deep learning has been developed [39] and microarchitectures of modern GPUs have begun to incorporate specialized computing units

for deep learning [58], indicating a strong and growing interest towards deep learning in both academia and industry alike.

Structure of Neural Networks

Neural networks can be thought of as being large function compositions. Each neuron that is part of the network acts as a single function, taking as input the weighted output of other neurons connected to it and outputting the value of the neuron's *activation function*. Neural networks can be classified into two main categories: feedforward and recurrent neural networks. Thinking of neurons and their connections in terms of graphs, where neurons represent vertices and edges represent the flow of data from one neuron to another, these two categories correspond to acyclic directed graphs and directed graphs that may contain cycles. In this work we will limit our focus to feedforward neural networks as the methods discussed do not make use of recurrent neural network structures.

As an example, Figure 2.1 depicts a feedforward neural network taking input vectors $\mathbf{x}$ of length 3 and outputting vectors $\mathbf{y}$ of length 2.[1] The network can be seen to be split into three functionally distinct parts: an input layer, hidden layers, and an output layer. The input layer is directly fed the values to be run through the network. The hidden layers are formed by neurons connecting the input to the output layer. The output layer represents the result of the computation performed by the network. What this depicted neural network computes can be expressed with the equation

$$\mathbf{y} = f_3(\mathbf{W}_3 f_2(\mathbf{W}_2 f_1(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3),$$

where f_i are the activation functions, $\mathbf{W}_i$ the matrices of weights, and $\mathbf{b}_i$ the biases of the ith layer. The purpose of an activation function is to alter incoming signals to induce nonlinearity into the network as well as impose constraints on the values that flow through specific neurons, enabling the learning of more complex functions with desired properties. Bias neurons enable the shifting of activation function outputs. Weights of the network control the signal strengths from one neuron to the other.

Learning in Neural Networks

A simplistic view of neural networks would be to consider them merely as powerful function approximators. The theoretical underpinning for this view is the universal approximation theorem [33], which states that under mild conditions on the activation function, neural networks with a single hidden layer can arbitrarily well approximate any continuous function.[2,3] However, the universal approximation theorem does not state anything about

[1] Note that bias neurons have been omitted from the figure for simplicity.

[2] The universal approximation theorem requires that the activation function is nonconstant, bounded and continuous, which highlights the importance of nonlinear activation functions.

[3] To be precise, this approximation applies to continuous functions defined on compact subsets of $\mathbb{R}^n$.

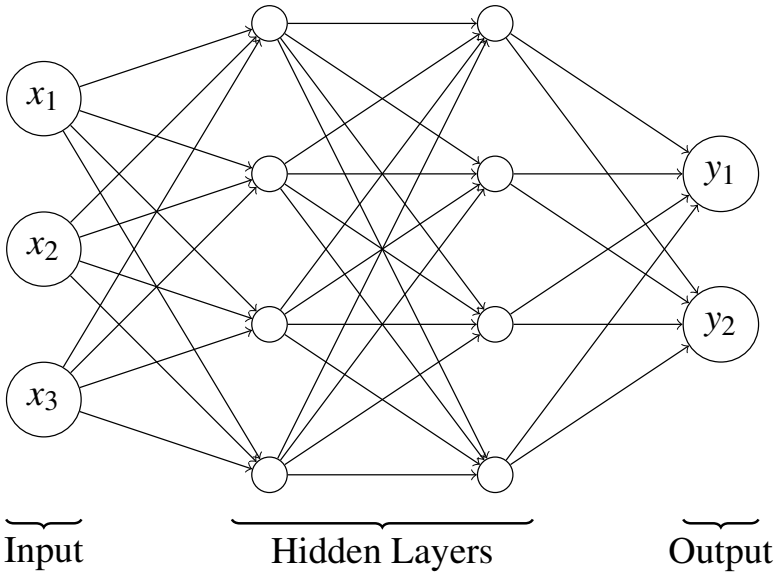

Figure 2.1. *Feedforward neural network with input dimension 3, output dimension 2 and 2 hidden layers of dimension 4 each.*

the challenges associated with the learnability of the parameters for these approximations, which is where many of the practical challenges associated with neural networks lie.

Learning in neural networks boils down to the efficient finding of correct weights for the network to produce desired outputs for given inputs. How well a given network performs is evaluated by computing the value of its *objective function*, given the knowledge of the input and output values. Finding weights for a given network thus becomes a problem of optimizing the value of its objective function. In this manner, neural networks can be trained for various computational tasks by choosing a suitable network architecture and specifying an objective function that models the computational task the network is desired to perform.

Early on in the development of neural networks the simple yet powerful backpropagation algorithm was developed independently by multiple researchers and later popularized by Rumelhart *et al.* [71] to enable the efficient training of neural networks. In essence, backpropagation is an algorithm to optimize neural network weights by exploiting their compositional structure and the chain rule of derivatives.

The backpropagation algorithm consists of two steps, the forward pass and the backward pass. In the forward pass input values are propagated through the network and the value of the objective function is computed. Following the forward pass, in the backward pass the gradient of the objective function is computed recursively w.r.t. all the weights of the network by beginning at the output layer and applying the chain rule to propagate the calculation back through the network. These computed gradients are then used to update the weights.

Various optimization methods can be used in conjunction with the backpropagation al-

gorithm to perform the weight updates w.r.t. the computed gradients. The gradients themselves give the direction and magnitude the weights of the network should be updated to maximize the value of the value of the objective function for the given input. [Note: On different optimization methods. Online/batch/stochastic GD, adaptive gradient methods, mention second order methods like [59]?]

As a simple example, we will consider the learning task of approximating a non-linear function with a neural network on a bounded domain. The function we aim to approximate is $f(x) = \sin x$ on the bounded domain $[0, 2\pi]$. As both the domain and range of the function being approximated are 1-dimensional, both the input and output layer are chosen to contain one node each, while 3 hidden layers of 10 nodes each with the tanh activation function applied to them are used. We train the neural network using standard SGD to optimize the *mean squared error* (MSE) objective:

$$\mathscr{L}_{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left\| \sin x^{(i)} - y^{(i)} \right\|^2,$$

where N corresponds to the batch size, $x^{(i)}$ the ith network input of the batch and $y^{(i)}$ the corresponding output. We choose the batch size to be $N = 100$ and run SGD for 2000 iterations, sampling each $x^{(i)}$ uniformly from the domain $[0, 2\pi]$. Figure 2.2 visualizes the output of the neural network at three different points of training, where the input points used are the same set of points uniformly sampled from the domain $[0, 2\pi]$. As more iterations of training are performed the neural network can be observed to converge to outputting the desired values.

The function we chose to approximate is trivial, seeing that instead of using several hidden layers with multiple nodes and a hyperbolic activation functions we could have chosen a much simpler network structure and simply used sin directly as an activation function. However, this example does serve to highlight the capability of a neural network to adapt its weights according to the objective even with the initial configuration being suboptimal.

Several difficulties still arise when attempting to utilize neural networks, e.g. how to avoid poor local minima, how to make the training converge faster, and how to avoid underfitting or overfitting. In recent years, both theoretical and empirical work has been conducted to better understand these phenomena as well as develop methods to combat the difficulties they impose on learning in neural networks. [Note: explain some of the recent research regarding convergence, vanishing gradients, adaptive momentum based methods, regularization, training deeper networks ... [36, 81, 23, 26, 18, 84, 42], and how there still isn't consensus on why some established methods seem to work ([74]) (learning rate/batch size choice for optimization [41, 79]) (potentially worse generalization of adaptive gradient methods, such as AdaGrad, RMSProp, Adam [92], even though empirical evidence suggests generally better performance on real-world data sets.) (initializing weights [22])]

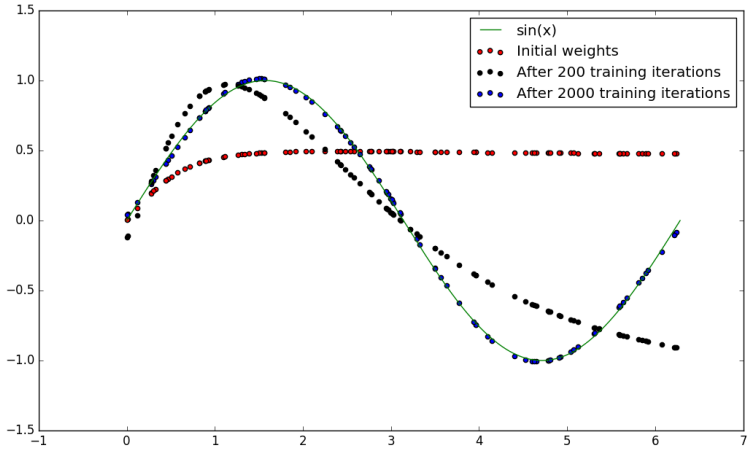

Figure 2.2. *Approximating* $\sin x$ *on the bounded domain* $[0, 2\pi]$ *with a neural network.*

2.1.2 Autoencoders

First discussed in [70] as "the encoding problem", autoencoders (AE) are a class of neural network architectures designed for unsupervised representation learning. In essence, unsupervised representation learning attempts to find relevant features of a given data set without additional input to guide the process. Development of such methods is of interest to bypass the need for manual feature engineering.

The structure of autoencoders can be decomposed into two parts, namely the encoder and decoder. As the constituents' names suggest, part of the network performs encoding of the input and the other decoding of the encoded input, hence the name of this neural network architecture as a whole referring to "self-encoding". An example autoencoder network has been visualized in Figure 2.3.

The learning objective of autoencoders is for the network to be able to reconstruct its inputs as faithfully as possible. A common objective function to train an autoencoder with is to have it minimize the MSE between its input and output:

$$\mathscr{L}_{AE-MSE} = \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}\right\|^2, \tag{2.1}$$

where $\mathbf{x}^{(i)}$ is the ith original input, $\hat{\mathbf{x}}^{(i)}$ the reconstruction obtained by first encoding and then decoding the input and N is the size of the data set. Through optimization of such reconstruction the network attempts to learn to produce an efficient encoding capable of capturing salient features of the data it has been trained on.

In a trivial case, where the network has too much capacity for modeling the input data, an AE can simply learn the identity mapping. As this is clearly not conducive to learning

meaningful latent representations of the data, constraints on either the network structure or penalization of trivial representations are employed. The AE in Figure 2.3 depicts a structural constraint that forces the input data to flow through a *bottleneck*, a layer with less capacity than the original input, forcing the network to learn a representation of the data that is lower dimensional than the original. The bottleneck layer output $\mathbf{z}$ is typically viewed as the encoding, or latent representation, produced by the AE. By forcing a compressed representation with the goal of being able to reconstruct the original data as faithfully as possible the network naturally learns to reject the least important features, such as noise, present in the data.

A connection between PCA and autoencoders which are constrained to only have linear activation functions and are optimized with the MSE objective (see Equation 2.1) has been shown in [4]. As such, autoencoders that are not constrained to linear activation functions and are optimized with the MSE objective can be thought of as a generalization of PCA, capable of producing arbitrarily complex non-linear mappings in addition to linear ones.

Autoencoders with multiple hidden layers were historically hard to optimize, as often gradient descent would get stuck in poor local minima. A method for initializing weights via stacking RBMs and fine-tuning with gradient descent first described in [32] showed the efficacy of deep learning for non-linear dimensionality reduction. Further subsequent advances in training deep neural networks, previously discussed in Section 2.1.1, have mostly done away with the need for such pretraining. In practice, autoencoders can be trained very efficiently with the use of modern deep learning techniques.

Even though AEs provide a powerful and scalable non-metric[4] approach to dimensionality reduction there are a few disadvantages inherent to how they learn representations. First, it is difficult to interpret what the latent representation models, since it is the result of a potentially highly complex non-linear transformation and the space of latent values is not guaranteed to have any particular structure. Secondly, it is unclear how well the model will generalize to unseen data as the model does not explicitly perform regularization, thus regularization must be imposed by means of *ad hoc* methods. To address these issues we will instead be considering a stochastic extension of the autoencoding framework in the upcoming parts of this work.

2.1.3 Information Theory

As a primer to variational inference, the main quantities of interest in information theory will be covered here, as they play a central role in all the subsequently discussed methods.

[4]Unlike most non-linear methods of dimensionality reduction, learning the lower dimensional representation with AEs does not involve preservation of pairwise distances of data points. This is an important property especially for very high dimensional data, as distance metrics break down due to the curse of dimensionality.

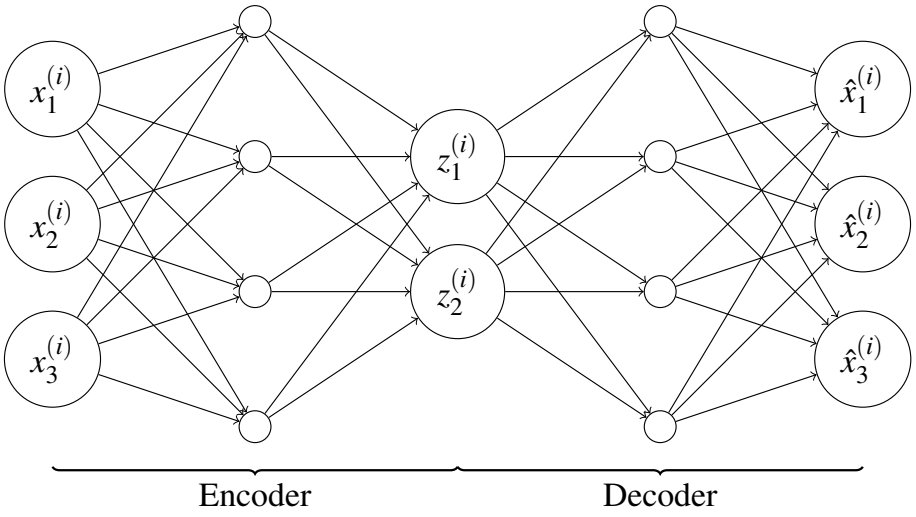

Figure 2.3. *Autoencoder with one hidden layer in both the encoder and decoder and latent dimensionality of 2.*

Information theory is the study of quantifying the information produced by a stochastic source.

[Note: Start from definition of self-information]

$$I_{p(\mathbf{x})}(\mathbf{x}) = -\log p(\mathbf{x})$$

The base of the logarithm determines the unit of information being considered, where popular choices are 2 for bits and e for nats. For convenience, we shall refer to bits as the unit of choice for the remainder of this section.

[Note: Entropy, as the expectation of self-information of a stochastic source.]

$$H(p) = \mathbb{E}_{p(\mathbf{x})}[I_{p(\mathbf{x})}(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x})}[-\log p(\mathbf{x})].$$

Using the formula defined above we can determine that the information entropy of a coin flip is 1 bit, whereas the entropy of a 6-sided die roll is approximately 2.58 bits.

[Note: Give encoding interpretation of entropy and include cross-entropy defined as:]

$$H(p,q) = \mathbb{E}_{p(\mathbf{x})}[-\log q(\mathbf{x})]$$

Relative entropy, also known as the Kullback-Leibler divergence, or KL-divergence for short, ties together the concepts of entropy and cross-entropy. [Note: Add the more practical explanation, which relates back to compression: $D_{\mathrm{KL}}(p \parallel q)$ determines the

average additional bits (or whichever unit is in use) needed for compression if q is used as a surrogate for p.]

[Note: define dkl in terms of entropy and cross entropy:]

$$D_{\mathrm{KL}}(p \parallel q) = H(p,q) - H(p) = \mathbb{E}_{p(\mathbf{x})}[-\log \frac{q(\mathbf{x})}{p(\mathbf{x})}]$$

A second interpretation for KL-divergence is that of a measure of dissimilarity between two probability distributions. As such, ...

The following properties hold for KL-divergence:

1. $D_{\mathrm{KL}}(p \parallel q) \geq 0$ (non-negativity)
2. $D_{\mathrm{KL}}(p \parallel q) = 0 \iff p = q$ (identity of indiscernibles).

One must note however that KL-divergence is asymmetric, i.e. $D_{\mathrm{KL}}(p \parallel q) \neq D_{\mathrm{KL}}(q \parallel p)$, and does not obey the triangle inequality, thus cannot readily be used as a metric. Especially asymmetry is important to take into account when interpreting the use of KL-divergence. Figure 2.4 visualizes the effect this asymmetry has as the KL-divergence between a Gaussian distribution q and a mixture of Gaussians distribution p is minimized.

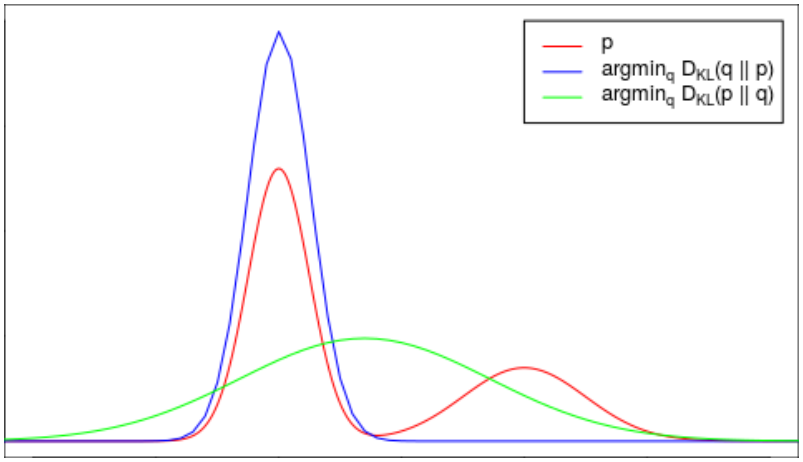

***Figure 2.4**. Asymmetry of KL-divergence as the KL-divergence between a Gaussian distribution q and a mixture of Gaussians p is minimized.*

2.1.4 Variational Inference

In modern statistics and machine learning a need to approximate highly complex probability distributions is a problem that often arises. A domain where such complex distributions are often encountered is Bayesian inference. The Bayes theorem states that the posterior distribution of latent variables conditioned on observed data is given by

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{z},\mathbf{x})}{\int p(\mathbf{z},\mathbf{x})d\mathbf{z}},$$

where the vector $\mathbf{x}$ denotes the observed variables and $\mathbf{z}$ the latent variables of the model. Even for only moderately complex models exact computation of the posterior is often intractable due to the required marginalization over all possible latent values $\mathbf{z}$.

To illustrate this problem we will consider the same Bayesian mixture of Gaussians model described in [6]. The model under consideration is univariate and consists of K unit variance Gaussian mixture components. The full hierarchical model is given by the following

$$\begin{aligned} \mu_k &\sim \mathcal{N}(0,\sigma^2), && k = 1,\ldots,K, \\ c_i &\sim \text{Categorical}(1/K,\ldots,1/K), && i = 1,\ldots,n, \\ x_i|c_i,\boldsymbol{\mu} &\sim \mathcal{N}(c_i^\top \boldsymbol{\mu},1), && i = 1,\ldots,n, \end{aligned}$$

where each of the n observations x_i are generated by drawing the means of the K Gaussian mixture components μ_k and the mixture component assignments c_i.[5] In this model the latent variables are $\mathbf{z} = \boldsymbol{\mu},\mathbf{c}$. From these definitions, the joint distribution factorizes as

$$p(\boldsymbol{\mu},\mathbf{c},\mathbf{x}) = p(\boldsymbol{\mu})\prod_{i=1}^{n} p(c_i)p(x_i|c_i,\boldsymbol{\mu}).$$

Now, when marginalizing out $\boldsymbol{\mu}$ and $\mathbf{c}$, we arrive at the following integral

$$p(\mathbf{x}) = \int p(\boldsymbol{\mu})\prod_{i=1}^{n}\sum_{c_i} p(c_i)p(x_i|c_i,\boldsymbol{\mu})d\boldsymbol{\mu}.$$

As the time complexity of evaluating this integral is $\mathcal{O}(K^n)$, it is evident that direct computation quickly becomes infeasible even for relatively small K and n.

The dominant paradigm for obtaining approximates for such distributions has been through various methods of sampling, such as Markov chain Monte Carlo (MCMC) methods like Gibbs sampling or the Metropolis-Hastings algorithm. [Note: These sampling methods provide asymptotic guarantees of exactness. More computationally demanding but more accurate approximates than VI. VI more suited to the large data sets prevalent in modern machine learning.]

[5] Note that c_i is being encoded as an indicator K-vector, where all entries are zero except for the index that corresponds to the drawn categorical variable being 1.

Variational inference takes a different approach to approximating the posterior. Instead of sampling, in variational inference the approximation problem is turned into an optimization problem. By considering a family of distributions $\mathscr{Q}$ one seeks to optimize

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in\mathscr{Q}}{\arg\min} D_{\mathrm{KL}}(q(\mathbf{z}) \parallel p(\mathbf{z}|\mathbf{x})), \tag{2.2}$$

where the quantity being optimized is the KL-divergence introduced in the previous section. The optimization tries to find a member $q^*(\mathbf{z})$ of the chosen family of distributions that best matches the true posterior, which can then be used as a surrogate for the true posterior.

In practice, the considered family of distributions $\mathscr{Q}$ is constrained in a way as to make optimization tractable. A common constraint is to assume the distributions of the chosen family factorize as follows

$$q(\mathbf{z}) = \prod_{i=1}^{d} q_i(z_i),$$

which is referred to as the *mean field approximation* in literature. The mean field approximation essentially removes from consideration all covariate structure as each variable of $\mathbf{z}$ is considered independent. In the Gaussian case this means instead of optimizing d^2 variables of the covariance matrix we can instead limit optimization to only consider d elements of a diagonal covariance matrix. An illustration of this for the case of two Gaussian distributions is presented in Figure 2.5.

Seeing that Equation 2.2 cannot be directly optimized as it depends on evaluation of the intractable posterior, in variational inference one seeks to instead optimize an alternative objective that is tractable. To derive this alternative objective, we begin with the definition

$$\begin{aligned} D_{\mathrm{KL}}(q(\mathbf{z}) \parallel p(\mathbf{z}|\mathbf{x})) &= \mathbb{E}[\log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}] \\ &= \mathbb{E}[\log q(\mathbf{z})] - \mathbb{E}[\log p(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}[\log q(\mathbf{z})] - \mathbb{E}[\log p(\mathbf{z},\mathbf{x})] + \log p(\mathbf{x}), \end{aligned} \tag{2.3}$$

and by rearranging, it follows from the non-negativity property of KL-divergence that

$$\begin{aligned} \log p(\mathbf{x}) &= D_{\mathrm{KL}}(q(\mathbf{z}) \parallel p(\mathbf{z}|\mathbf{x})) - \mathbb{E}[\log q(\mathbf{z})] + \mathbb{E}[\log p(\mathbf{z},\mathbf{x})] \\ &= \underbrace{D_{\mathrm{KL}}(q(\mathbf{z}) \parallel p(\mathbf{z}|\mathbf{x}))}_{\geq 0} + \mathbb{E}[\log p(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q(\mathbf{z}) \parallel p(\mathbf{z})) \\ &\geq \underbrace{\mathbb{E}[\log p(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q(\mathbf{z}) \parallel p(\mathbf{z}))}_{\text{ELBO}}. \end{aligned} \tag{2.4}$$

This bound is better known as the *evidence lower bound*, or ELBO for short. By maximizing this bound the optimization indirectly minimizes the KL-divergence between $q(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$, which corresponds to the original objective of variational inference.

[Note: It is possible to combine automatic differentiation with variational inference to further ease statistical model building and computational efficiency [48]. (Brief discussion on probabilistic programming)]

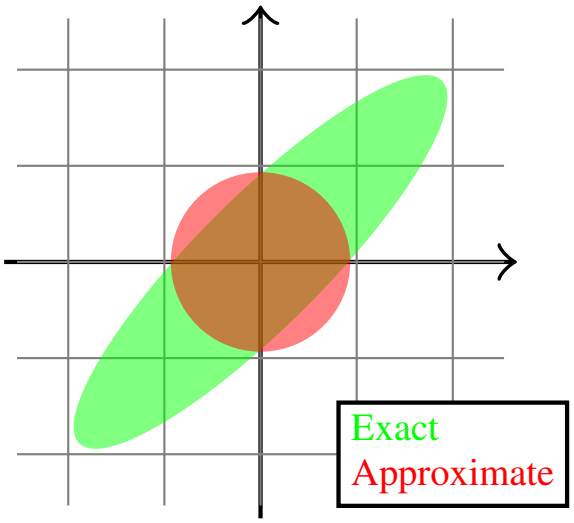

Figure 2.5. *Illustration of the mean-field approximation of a full rank Gaussian. Adapted from [6], Figure 1.*

2.1.5 Autoencoding Variational Bayes

VAEs extend autoencoders into probabilistic generative models. Instead of the deterministic encoding and decoding networks of traditional autoencoders, VAE encoders and decoders parameterize probability distributions. The encoder network parameterizes the distribution of the latent variable $\mathbf{z}$ conditioned on its data $\mathbf{x}$, whereas the decoder parameterizes the data distribution conditioned on a latent variable.

[Note: Explain the relationship between latent variables, their distributions and why variational bayes is needed here. ... In terms of probability distributions the generative model $p_\theta(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$ is parameterized by θ.]

To summarize, Figure 2.6 depicts the graphical model under consideration, where ϕ and θ are the parameters of the encoder and decoder, respectively.

Derivation of the VAE Objective Function

[Note: TODO: Now there's overlap here with the VI section. Reword this as an example of it, rather than introducing the whole concept of ELBO again.]

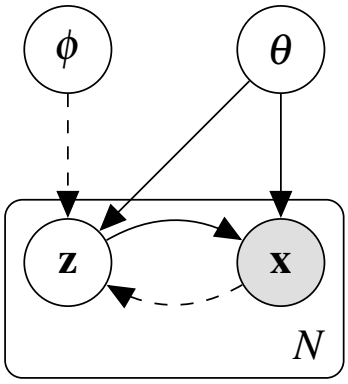

Figure 2.6. *The VAE graphical model, in plate notation. Adapted from [45], Figure 1.*

The objective of VAEs is to model the generative process of a given data set as well as possible. The objective is derived from the maximization of the marginal likelihood $p_\theta(\mathbf{x}^{(i)})$ of each data point i in the given data set. By application of Jensen's inequality we can derive the following lower bound for the marginal likelihood for our latent variable model:

$$\begin{aligned}
\log p_\theta(\mathbf{x}^{(i)}) &= \log \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z} \\
&= \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z} \\
&= \log \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\frac{p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\right] \\
&\geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right] \\
&= -D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right].
\end{aligned} \tag{2.5}$$

This bound is better known as the *evidence lower bound*, or ELBO for short, which we will denote as $\mathscr{L}(\theta, \phi; \mathbf{x}^{(i)})$. An interpretation for the terms of the last equation is that the objective is to have the posterior $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ match the prior $p_\theta(\mathbf{z})$ as accurately as possible, acting as a regularizer, while simultaneously maximizing the expectation of the likelihood of $\mathbf{x}^{(i)}$ under the learnt latent distribution, i.e. having the decoder parameterize a distribution from which a draw of $\mathbf{x}^{(i)}$ is as likely as possible.

Several modifications to this objective have been proposed, e.g. the β-VAE [28, 8], β-TCVAE [11] and InfoVAE [96], which aim to improve the learning of better latent representations. Beta-VAE prefixes the KL term with a constant β which is interpreted as a Lagrange multiplier that adjusts the regularization imposed by the latent prior. The case where $\beta = 1$ corresponds to the VAE formulation presented in this section, while $\beta > 1$ [Note: more learning pressure to match the prior, for factorized models this corresponds to more "disentanglement". β-TCVAE decomposes the β-VAE loss for factorized models to

reveal a correlation term between latent variables which it attempts to penalize. InfoVAE on the other hand attempts to maximize the mutual information between the latent code and the reconstructed output to enforce utilization of the latent code even when flexible decoder distributions are used.]

Obtaining Differentiable Monte Carlo Estimates

To optimize the parameters θ and ϕ of our neural network using backpropagation we require a way to compute the gradient of the expectations of random variables. The *reparameterization trick* enables keeping all computation the neural network performs fully deterministic while still permitting the sampling of probability distributions parameterized by the network itself. By keeping all computation deterministic, backpropagation can be used to optimize the network.

The reparameterization trick works by introducing an auxiliary noise variable as input to the network and a suitable differentiable function is used to transform the input noise into a sample of the distribution parameterized by the neural network and the original input data. As in [45], we shall denote this function by g_ϕ, the latent variable samples are then obtained by

$$\mathbf{z}^{(i,l)} = g_\phi(\mathbf{x}^{(i)}, \boldsymbol{\varepsilon}^{(l)}), \quad \boldsymbol{\varepsilon}^{(l)} \sim p(\boldsymbol{\varepsilon}).$$

The choice of distribution from which $\boldsymbol{\varepsilon}^{(l)}$ is drawn from and the analytic form of g_ϕ depends on the distribution that is being reparameterized. For location-scale distributions ...

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[f(\mathbf{z})] \simeq \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(\mathbf{x}^{(i)}, \boldsymbol{\varepsilon}^{(l)}))$$

$$\mathscr{L}(\theta, \phi; \mathbf{x}^{(i)}) = -D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p_\theta(\mathbf{z})) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) \tag{2.6}$$

In many cases the KL-divergence can be integrated analytically, for instance in the case of two Gaussian distributions. In cases where the KL term cannot be solved analytically a Monte Carlo estimate for the KL-divergence can be used:

$$\mathscr{L}(\theta, \phi; \mathbf{x}^{(i)}) = \frac{1}{L}\sum_{l=1}^{L} \left[q_\phi(\mathbf{z}^{(i,l)}|\mathbf{x}^{(i)}) - p(\mathbf{z}^{(i,l)}) + \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) \right]. \tag{2.7}$$

[Note: Should mention inability to reparameterize discrete distributions. A method for obtaining samples of a categorical distribution by relaxation to a continuous distribution [37], can avoid marginalizing over all possible values of latent categorical variables in models like the M2-VAE introduced in [43].] A drawback of the reparameterization trick is that it is only applicable to continuous distributions...

Network Structure

Figure 2.7 depicts the network structure of a VAE whose encoder outputs the mean and standard deviation parameters of a distribution. These parameters could for instance be used to model the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ as the multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}\mathbf{I})$, where, with slight abuse of notation, $\boldsymbol{\sigma}^{(i)}\mathbf{I}$ stands for a diagonal covariance matrix whose diagonal is made up of the elements of the vector $\boldsymbol{\sigma}^{(i)}$.

It is important to note that the output $\hat{\mathbf{x}}^{(i,l)}$ of the network in the figure should not be interpreted as the final reconstruction of the input like in traditional autoencoders, but rather as the values parameterizing the distribution $p_\theta(\mathbf{x}|\mathbf{z}^{(i,l)})$. For binary data, the distribution $p_\theta(\mathbf{x}|\mathbf{z}^{(i,l)})$ is commonly chosen to be a multivariate Bernoulli, where the decoder outputs $\hat{\mathbf{x}}^{(i,l)}$ parameterize individual Bernoulli distributions, whereas for continuous data the parameters for a Gaussian distribution are typically outputted.

[Note: Problem with variational inference, also apparent in our Gaussian encoder example here, is the inflexibility of the chosen distribution. Normalizing flows attempt to resolve this issue [67, 44, 86], result in overall better $p(\mathbf{x})$, and with VAE these flows can be learned efficiently.]

[Note: Extensions to semi supervised and adversarial learning [43, 57].]

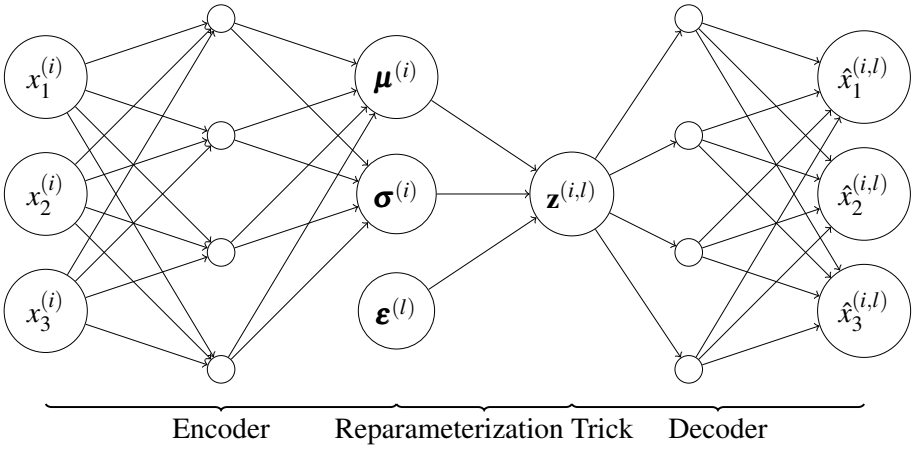

Figure 2.7. *Variational Autoencoder. Note that the hidden layer values $\boldsymbol{\mu}^{(i)}$, $\boldsymbol{\sigma}^{(i)}$ and the input noise variable $\boldsymbol{\varepsilon}^{(l)}$ represent vectors instead of single values for a cleaner visual representation.*

2.2 t-Distributed Stochastic Neighbor Embedding

Taking a slight departure from the preceding discussion on neural networks, the theory, and development leading to the acclaimed dimensionality reduction algorithm t-SNE will be presented. At the end of this section, a way to utilize the t-SNE objective to train an embedding function parameterized by a neural network will be shown, thereby connecting this section with the previous in a natural way, as well as covering the final preliminaries to the contribution of this thesis.

2.2.1 SNE

The Stochastic Neighbor Embedding (SNE) algorithm [31], developed by Hinton and Roweis in 2003, serves as the precursor to the later improved t-SNE algorithm. As such, SNE shares many similarities in its definition with t-SNE, and thus by first developing an understanding of SNE and its pitfalls the motivation behind the t-SNE algorithm become more apparent.

As the algorithm's name suggests, the approach SNE takes to perform dimensionality reduction is probabilistic in nature. Without delving directly into the mathematical details, an intuitive overview of the algorithm is that it attempts to embed data points in a lower dimension by preserving probability distributions defined on the pairwise distances of points. More concretely, the algorithm proceeds as follows. First, for all data points probability distributions for how likely it is to pick each point as its neighbor in both input and output space are formed. After these probability distributions have been formed, the actual embedding is performed by attempting to distribute the points in the output space such that similarly defined probability distributions in the output space match those of the input space as closely as possible. The natural way to match two distributions is familiar from the discussion on variational bayes methods, namely the KL-divergence between the input and output space distributions is used as the objective function for the embedding of data points, which can be optimized with various methods, including the previously covered gradient descent.

To formalize these notions, we begin with how the conditional probability of data point i picking data point j in the input space is defined by the equation

$$p_{j|i} = \frac{exp(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} exp(-\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2 / 2\sigma_i^2)}, \tag{2.8}$$

where the probability can be seen as being derived as if a Gaussian were centered on $\mathbf{x}_i$ and an appropriate normalizing term is used to scale the individual probabilities so that they sum to 1. The variance (denoted by σ_i in the equation) is determined separately for

each data point i by forcing the perplexity of the conditional distribution over all other data points (later denoted as P_i) to equal some preset value.

Perplexity is an information theoretic measure of how well a distribution predicts its samples. For a given probability distribution, perplexity is defined as the exponentiation of the distribution's entropy H, e.g. in the case of our discrete P_i:

$$b^{H(P_i)} = b^{-\sum_{j=1}^{n} p_{j|i} \log_b p_{j|i}},$$

where b is the unit used to measure entropy, typically chosen to be either 2 or e, for bits or nats, respectively. From the definition, we can see that larger variance will be chosen for the Gaussians in areas of the input space where the density of data points is low and smaller variance will be used in high density areas. Effectively, variance is adapted to fit the core assumption that the data points lie on a uniformly sampled manifold within the input space. Perplexity is thus a value proportional to the number of effective local neighbors considered by the algorithm when the embedding is being optimized. In practice, the correct σ_i for each i is numerically sought using the bisection method, up to a small error tolerance in the resulting perplexity.

The conditional distribution in the output space is similarly defined with a Gaussian kernel:

$$q_{j|i} = \frac{exp(-\left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2)}{\sum_{k \neq i} exp(-\left\|\mathbf{y}_i - \mathbf{y}_k\right\|^2)},$$

where each $\mathbf{y}_i$ represents the point in the embedding for data point $\mathbf{x}_i$. A notable difference is the absence of the variance term $2\sigma_i^2$, which can be interpreted as having been set to 1 throughout. This choice of fixed variance can, in turn, be interpreted as smoothing out the manifold of the input space when performing the embedding.

Now that we have defined both the output and input distributions for points, the objective function can be formulated as the sum of KL-divergences between each input-output distribution pair:

$$C_{\text{SNE}} = \sum_i D_{\text{KL}}(P_i \parallel Q_i) = \sum_i \sum_{j \neq i} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}. \tag{2.9}$$

To optimize this objective, gradient descent is typically used to update the positions of the embedded data points. The partial derivative of the objective w.r.t. a point i in the embedding is given by

$$\frac{\partial C_{\text{SNE}}}{\partial \mathbf{y}_i} = 2 \sum_{j \neq i} (\mathbf{y}_i - \mathbf{y}_j)(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}), \tag{2.10}$$

which can be used to form the gradient. This has the intuitive interpretation of a force directed layouting algorithm, where each pair of points is either pushing or pulling on each other, depending on whether they are less or more likely to be neighbors under the output's conditional distribution than the input's.

An important fact to note is that the objective function is in fact non-convex, which implies the unfortunate property that the optimization of the objective function is not guaranteed to terminate at the global optimum due to the potential existence of multiple local minima. One of the main drawbacks of SNE is how it easily ends up in poor local minima. A common way to attempt to avoid poor local minima is to add Gaussian noise to the points in the embedding which is gradually reduced during optimization. This however adds more tunable hyperparameters to the algorithm. In addition to adding this jitter to the embedding points, random restarts are recommended in practice, which become very costly to run when dealing with large data sets.

2.2.2 t-SNE

The t-SNE algorithm aims to address two issues of the original SNE algorithm. First, one of the main issues with plain SNE is the so-called *crowding problem*, which is addressed by considering a heavier tailed probability distribution function in the output space, namely the Student's t-Distribution. Second, through considering joint distributions instead of conditional distributions and the use of a different distribution in the output space lead to an objective function that is easier to optimize than that of SNE. Together these improvements enable better separation of natural clusters in the output space of the algorithm, as compared to the original SNE method.

The crowding problem is named after the phenomenon where distances in a high dimensional space cannot faithfully be modelled in a lower dimensional space, causing the SNE algorithm to collapse points together in the embedding and thus preventing the formation of natural clusters. One can easily verify that it is not possible in the general case to preserve distances perfectly when performing dimensionality reduction, as a space of n dimensions allows for a maximum of $n+1$ equidistant points. How the crowding problem manifests can be seen by considering how the volumes of objects, and by extension the densities of probability distributions, are concentrated in high dimensional space. Unintuitively, as the number of dimensions increase, the concentration of a Gaussian distribution's density shifts from its mean to a shell around its mean [90, p. 50]. When SNE attempts to match probabilities derived from Gaussian kernels in different dimensionalities, moderately distant points in the high dimensional space will have a disproportionately large force attracting them to the center of the lower dimensional kernel. In addition to SNE, the crowding problem is also present in other nonlinear dimensionality reduction methods which are based on preservation of pairwise distances of points, such as Sammon mapping [54].

To address the crowding problem the t-SNE algorithm switches the Gaussian kernels of the output space to ones formed with the Student's t-Distribution. The benefit a t-Distribution

has over a Gaussian is that it places more of its probability density in its tails. This property of the distribution allows moderate distances in the input space to be modeled further apart in the output space, alleviating the crowding problem by reducing the attractive forces between moderately distant points and thereby facilitating the formation of more distinct clusters in the embedding. In addition, using a t-Distribution has the computational advantage of not having to perform exponentiation when computing probabilities.

The probability density function of a t-Distribution with ν degrees of freedom is given by

$$f(x|\nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\,\Gamma(\frac{\nu}{2})}(1+\frac{x^2}{\nu})^{-\frac{\nu+1}{2}},$$

which can be recognized to be closely related to the Gaussian distribution[6], in fact as $\nu \to \infty$ the probability density approaches that of the standard normal distribution. One can observe that the number of degrees of freedom controls how much probability density is placed in the tails of the distribution. The standard choice for t-SNE is to choose $\nu = 1$ when reducing to a very low number of dimensions, such as 2 or 3 for visualization of a data set as a scatter plot. However, more generally it is suggested to consider higher degrees of freedom when embedding to higher dimensional spaces, as the crowding problem becomes less prominent and thus the required amount of compensation is reduced.

Similarly to Symmetric SNE [13], instead of matching conditional distributions in the input and output spaces, t-SNE forms a joint distribution over all pairs of points in both the input and output space, which it attempts to match. The joint distribution over the input points (P) is formed by using Equation 2.8:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N},$$

and by using the t-Distribution in place of a Gaussian, the output space joint distribution becomes

$$q_{ij} = \frac{(1+\left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2/\nu)^{-\frac{\nu+1}{2}}}{\sum_{k\neq l}(1+\left\|\mathbf{y}_k - \mathbf{y}_l\right\|^2/\nu)^{-\frac{\nu+1}{2}}}.$$

The t-SNE objective function is defined similarly to the SNE objective (cf. Equation 2.9), except the sum is taken over KL-divergences of the joint probabilities:

$$C_{\text{t-SNE}} = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

[6] From a statistical perspective, the t-distribution arises when estimating the mean of finite Gaussian data with unknown variance.

For which the following gradient can be derived:

$$\frac{\partial C_{\text{t-SNE}}}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (\mathbf{y}_i - \mathbf{y}_j)(p_{ij} - q_{ij})(1 + \left\| \mathbf{y}_i - \mathbf{y}_j \right\|^2 / \nu)^{-\frac{\nu+1}{2}}, \tag{2.11}$$

from which we can observe an additional benefit of choosing $\nu = 1$: to evaluate the gradient with $\nu = 1$, only elementary arithmetic operations need to be computed, decreasing the computation time of the optimization. Methods originally developed for improving the runtime performance of n-body simulations have been adapted to speed up the optimization of t-SNE [95, 53]. Use of the fast multipole method [69] however is not feasible due to no suitable multipole expansion for the t-Distributions in the output space exist. More discussion on these methods and performance considerations w.r.t. our method is presented in Section 4.4.2.

Further variations of the t-SNE method, beyond performance enhancements to the original, have been proposed. One of particular interest is the extension of t-SNE to a parametric dimensionality reduction method [52], which will be covered in the next section, as it is a central component to the method presented in this work. Other methods worth a mention include Stochastic Triplet Embedding [87] and Multiple Maps t-SNE [55], developed to handle non-metric similarities present in the data.

2.2.3 Parametric t-SNE

A major drawback of most nonlinear dimensionality reduction methods is that they do not directly enable out-of-sample extensions. A natural extension to t-SNE is to learn a function parameterizing the embedding from input space to output space, i.e. instead of optimizing the t-SNE objective by simply moving a fixed set of points in the output space, we would instead optimize the parameters of a function that maps training data points from the input space to the output space. Once parameters for such a function are found the function can be used to not only embed the training data but any subsequently collected data points that belong to approximately the same manifold.

The basic idea behind parametric t-SNE is straightforward. A feedforward neural network is used to parameterize a function $f_{\mathbf{W}} : X \mapsto Y$, which maps points from input space to output space in such a way as to minimize the t-SNE objective. Once the weights of the neural network are learned, performing dimensionality reduction on any given point is done simply by running it through the neural network. This parametric extension to t-SNE was first realized by van der Maaten [52], and we will henceforth refer to this method as PTSNE.

By application of the chain rule, we can write the gradient w.r.t. the parameters of the neural network as

$$\nabla_{\mathbf{W}} C_{\text{t-SNE}} = \frac{\partial C_{\text{t-SNE}}}{\partial \mathbf{W}} = \frac{\partial C_{\text{t-SNE}}}{\partial f_{\mathbf{W}}(\mathbf{x}_i)} \frac{\partial f_{\mathbf{W}}(\mathbf{x}_i)}{\partial \mathbf{W}}.$$

As $f_{\mathbf{W}}(\mathbf{x}_i) = \mathbf{y}_i$, Equation 2.11 is used to compute the first term of the product, while standard backpropagation can be applied to the rest of the network.

3. METHOD

Scalable, non-linear dimensionality reduction methods that are robust to the curse of dimensionality while being able to produce meaningful visualizations for exploratory data analysis [Note: ".. don't really exist .."]. Even though VAEs address issues with scalability and reliance on distance metrics, the resulting latent space representations do not allow for as good separation of clusters as other methods, such as t-SNE or UMAP [60], which will later be demonstrated in Section 4.4.2. On the other hand, PTSNE is in theory capable of achieving embeddings that are comparable to t-SNE, but in practice, the optimization often results in much noisier embeddings, where for instance cluster boundaries are not as clearly visible.

The method proposed in this work involves combining the VAE model with PTSNE to produce a general framework for improved parametric dimensionality reduction for data visualization. By combining the two models in a way where the strengths of both methods compliment each other the weaknesses of each individual method are diminished.

In this chapter the combined model, its training procedure, its properties, and a variation of the method are discussed.

3.1 Learning a Parametric Embedding Using VAE Sampling

As the aim of the method is to take advantage of the strengths of both VAE and PTSNE, we begin by recapitulating the strengths of each method.

[Note: Pros of VAE: generative model, stochasticity brings robustness, scalable parametric nonlinear dimensionality reduction] [Note: Pros of PTSNE: theoretically capable of learning as good projections as t-SNE, which is widely considered as the best option for data visualization. Also scalable and parametric, if trainable with smaller batch sizes.]

Using the notation introduced in Chapter 2, we can summarize the learning procedure as Algorithm 1. In essence, training the model is done in two phases...

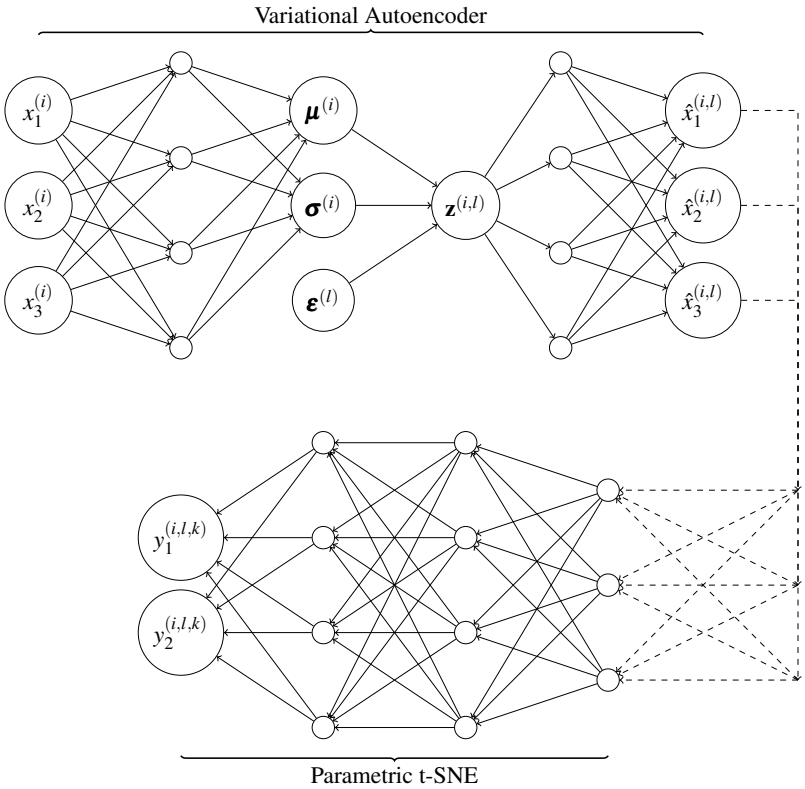

Figure 3.1. *Illustration of the general structure of a VPTSNE network.*

Algorithm 1 Training of the neural networks.

Input: Data set $\mathbf{D}$ of size N, VAE training batch size M_{VAE}, VAE samples L, PTSNE training batch size M_{PTSNE}

Output: Neural network parameters $\theta, \phi, \mathbf{W}$

$\theta, \phi, \mathbf{W} \leftarrow$ Initialize neural network parameters
while VAE not converged **do**
  $\mathbf{X} \leftarrow$ Random minibatch of size M_{VAE} from $\mathbf{D}$
  for all $\mathbf{x}^{(i)} \in \mathbf{X}$ **do**
    for $l \leftarrow 1$ to L **do**
      $\boldsymbol{\varepsilon}^{(l)} \leftarrow$ Draw sample from noise distribution $p(\boldsymbol{\varepsilon})$
      $\mathbf{z}^{(i,l)} \leftarrow g_\phi(\mathbf{x}^{(i)}, \boldsymbol{\varepsilon}^{(l)})$ ▷ Draw from $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$
      $\hat{\mathbf{x}}^{(i,l)} \leftarrow f_\theta(\mathbf{z}^{(i,l)})$ ▷ Obtain parameters of $p_\theta(\mathbf{x}|\mathbf{z}^{(i,l)})$ by decoding $\mathbf{z}^{(i,l)}$
    end for
    $\mathscr{L}(\theta, \phi; \mathbf{x}^{(i)}) \leftarrow$ Compute estimate with either equation 2.6 or 2.7
  end for
  $\theta, \phi \leftarrow$ Updata parameters by ascending gradient $\nabla_{\theta,\phi} \frac{N}{M_{\mathsf{VAE}}} \sum_{i=1}^{M_{\mathsf{VAE}}} \mathscr{L}(\theta, \phi; \mathbf{x}^{(i)})$
end while
while PTSNE not converged **do**
  $\mathbf{X} \leftarrow$ Random minibatch of size M_{PTSNE} from $\mathbf{D}$
  for all $\mathbf{x}^{(i)} \in \mathbf{X}$ **do**
    $\boldsymbol{\varepsilon}^{(l)} \leftarrow$ Draw sample from noise distribution $p(\boldsymbol{\varepsilon})$
    $\mathbf{z}^{(i,l)} \leftarrow g_\phi(\mathbf{x}^{(i)}, \boldsymbol{\varepsilon}^{(l)})$ ▷ Draw from $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$
    $\widetilde{\mathbf{x}}^{(i)} \leftarrow$ Draw sample from $p_\theta(\mathbf{x}|\mathbf{z}^{(i,l)})$ ▷ Obtain reconstruction from VAE
    $\mathbf{y}^{(i)} \leftarrow f_{\mathbf{W}}(\widetilde{\mathbf{x}}^{(i)})$
  end for
  $\mathbf{W} \leftarrow$ Update parameters by descending gradient $\nabla_{\mathbf{W}} C_{\text{t-SNE}}$ using $\widetilde{\mathbf{x}}^{(i)}$ and $\mathbf{y}^{(i)}$

[Note: As we can draw infinite samples from the continuous distribution $p_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ with $g_\phi(\mathbf{x}^{(i)}, \boldsymbol{\varepsilon}^{(l)})$ ...]

After training, a deterministic projection for all subsequent points can be obtained with Algorithm 2. For encoder and decoder network pairs that both directly parameterize the mean of a distribution, computing the expectation can be done exactly with zero overhead.

Algorithm 2 Deterministic projection.

Input: Point $\mathbf{x}^{(i)}$, neural network parameters $\theta, \phi, \mathbf{W}$
Output: Projected point $\mathbf{y}^{(i)}$

1: $\widetilde{\mathbf{z}}^{(i)} \leftarrow \mathbb{E}[q_\phi(\mathbf{z}|\mathbf{x}^{(i)})]$
2: $\widetilde{\mathbf{x}}^{(i)} \leftarrow \mathbb{E}[p_\theta(\mathbf{x}|\widetilde{\mathbf{z}}^{(i)})]$
3: $\mathbf{y}^{(i)} \leftarrow f_{\mathbf{W}}(\widetilde{\mathbf{x}}^{(i)})$

3.2 Obtaining Reconstructions from Hidden Layers

Instead of training the embedding network on the final output of the VAE it is possible to instead use the outputs of a chosen hidden layer. If the chosen hidden layer has dimensions considerably smaller than that of the original, the benefits of this approach are twofold:

- We are performing a step of nonlinear dimensionality reduction that is not dependent on distance metrics, i.e. a preprocessing step that is less susceptible to the curse of dimensionality.
- Given that the number of dimensions can be chosen to be considerably lower than in the original input space, computing the t-SNE loss also becomes proportionally cheaper.

When considering precomputed P_{ij} matrices for each batch the gain in computational advantage does however diminish significantly, although the following caveats apply to the precomputation. Storing the precomputed matrices will require considerable space for large batch sizes and data sets. A way to circumvent the need for large storage space is to employ a k-NN approximation when computing P_{ij}, i.e. only considering a small number of neighbors when computing the conditional probabilities for each pair of points, which will result in a sparse matrix. This however increases the error in the computed gradient and relies on an efficient k-NN algorithm in practice, which require the use of approximation algorithms when the data's dimensionality is high, as the use of space partitioning data structures for exact k-NN becomes computationally intractable. Additionally, fixing the batches in advance hinders the performance of SGD as the gradient estimates become biased. Moreover, relying on precomputed P_{ij} matrices for fixed batches prevents the use of infinite sample generation from the VAE.

3.3 Inference with the Generative Model

From the underlying generative model we can estimate the marginal likelihood $p(\mathbf{x})$ for each data point $\mathbf{x}$. An estimation can be retrieved by importance sampling [68]:

$$p(\mathbf{x}) \simeq \frac{1}{N} \sum_{i=1}^{N} \frac{p(\mathbf{x}|\mathbf{z}^{(i)}) p(\mathbf{z}^{(i)})}{q_\theta(\mathbf{z}^{(i)}|\mathbf{x})},$$

where $\mathbf{z}^{(i)} \sim q_\theta(\mathbf{z}|\mathbf{x})$. Alternatively, one could directly use the ELBO to estimate $p(\mathbf{x})$; however, this gives generally worse estimates [93]. For more accurate estimates than importance sampling, the usage of annealed importance sampling [63] has been suggested for VAEs in [93].

Using the obtained marginals as feedback we can inform the user of potentially anomalous points in the data set. This feedback can then further be used in downstream analysis. From a data visualization perspective, points that do not fit the model well correspond to points that are likely to be mapped poorly, thus removing such outliers from the final mapping serves to highlight the natural clusters in the data more clearly.

3.4 Robustness to Sparse and Noisy Data

[Note: The unknown intrinsic dimensionality of the data manifold is reduced to at most the dimensionality of the latent code. Data less sparsely distributed on this manifold.] [Note: The stochastic model allows sampling infinitely many points on the learned manifold.]

As real-world data is generally not perfectly clean, an important property for any machine learning method is its capability to handle data containing artifacts. Considering VAEs are inherently robust to corrupted data due to the regularization provided by the stochastic latent code ([connection with VAE and Robust PCA [9, 10] shown in [14]]) it is reasonable to hypothesize that a training procedure for a low dimensional mapping taking advantage of this property would perform better on corrupted data.

3.5 Implementation

An accompanying implementation of the method has been made available at https://github.com/ahie/vptsne. The implementation is built on top of TensorFlow [1] and takes advantage of the TensorFlow Distributions [16] library. To take full advantage of the dataflow architecture of TensorFlow and the paralellism provided by GPUs, both the t-SNE loss and its gradient computation were implemented as custom CUDA [64] kernel operations.

The implementation has been made as extensible as possible by allowing the full specification of the neural network structures parameterizing the VAE distributions and the PTSNE

network. Utility functions for easily specifying common network structures are included in the implementation. The use of normalizing flows in the VAE model has also been made possible. Additionally, an implementation of the fast approximate nearest neighbor algorithm NN-descent [17] has been included in the t-SNE CUDA routines as an optional performance enhancement for computing the input space distribution P_{ij} when training with large batch sizes.

4. EXPERIMENTS

Several experiments are carried out to validate the efficacy of the method outlined in this thesis. The learning of a parametric embedding using our method is compared to plain PTSNE. Other methods of dimensionality reduction are compared against our method in terms of both embedding quality and scalability. Additionally, robustness properties of the method are demonstrated empirically as well as training the embedding on hidden layer outputs is experimented with. Lastly, two applications of the method to single cell data analysis are presented in section 4.5, including a demonstration of the use of the VAE for inference.

Before going into the results and applications of the method this chapter begins by first presenting the different data sets that will be used throughout the experiments after which the different quantitative metrics for evaluating the experimental results are covered and the general experimental setup, in regards to the used network structures, parameters and chosen optimization methods are discussed.

4.1 Data Sets

The five different data sets described below were used to conduct the experiments of this section.

4.1.1 MNIST

The MNIST data set [50] contains labeled images of handwritten digits. Each image contains a single digit in gray-scale with a resolution of 28x28 pixels. The data set is split into 60000 training images and 10000 test images. All networks were trained solely on the training data set, while the remaining test examples were reserved for assessing the capability of the network to generalize to out-of-sample extensions. In the unsupervised setting considered here, the provided labels were only used for visualizing the embeddings and evaluating the embedding quality of out-of-sample extensions via a 1-NN classifier.

MNIST is a standard data set that frequently appears in machine learning literature and is regularly included in unsupervised learning experiments. As a relatively simple labeled real-world data set it functions as a good baseline for studying the output of our method.

4.1.2 Fashion-MNIST

The Fashion-MNIST data set [94] was designed as a drop-in replacement for the MNIST data set and thus contains the same number of grayscale training and test images of equal

dimensions. The images themselves consist of 10 different classes of fashion items, such as T-shirts, trousers, sneakers, and handbags. The motivation for considering this data set in addition to MNIST is the fact that it has been shown to be harder to learn even with supervised computer vision methods.

4.1.3 Mass Cytometry

Cytometry is the measurement of biological characteristics of cells. In mass cytometry, time-of-flight mass spectrometry is used to measure the counts of cellular proteins present within a single cell by tagging proteins with their corresponding antibodies that have been conjugated with specific heavy metals.

As a comparatively new methodological development, mass cytometry enables simultaneous measurement of a considerably greater number of features compared to fluorescence-based flow cytometry. A greater number of features, however, poses new challenges in analyzing the measurement results. To aid the analysis of this high dimensional data it is common to visualize the data as a 2-D or 3-D scatter plot using t-SNE, due to its capability to separate biologically relevant subpopulations of cells in the produced embedding [3]. However, as a high-throughput method, mass cytometry data sets under analysis can grow to be millions of data points in size, prompting the use of more scalable algorithms. For a more thorough review of mass cytometry and associated analytical challenges, the reader is referred to [80].

In our experiments, we consider two different mass cytometry data sets. As the first data set, henceforth referred to as Cytometry A, we use the publicly available data of [51], which has been used to demonstrate the effectiveness of the Phenograph clustering algorithm on mass cytometry data. The data set consists of 81000 data points of 13 dimensions corresponding to the normalized counts of cell surface proteins belonging to distinct clusters of differentiation. As a benchmark data set for the Phenograph clustering algorithm, we additionally include the labels produced by Phenograph in our analysis.

The second mass cytometry data set (Cytometry B) contains data gathered from ovarian cancer patients at different phases of treatment. The number of points in the entire data set is 1.4 million, each of which has 23 dimensions.

4.1.4 Single-Cell RNA Sequencing

Single-cell RNA sequencing (scRNA-seq) enables the measurement of the transcriptional state of single cells. By mapping sequenced RNA fragments present in a given cell to RNA sequences of predetermined genes the expression profile of tens of thousands of genes can be measured simultaneously with current high-throughput sequencing methods. As the number of genes for which the strength of expression can be quantified is far greater than the number of proteins that can be measured with mass cytometry the computational and analytical challenges are further exacerbated [12, 82].

We explore how our method applied to scRNA-seq data reveals biologically meaningful insights by applying it to the melanoma patient data set studied in [85]. The data set consists of tumor samples from 19 different patients, altogether containing 4097 cells for which the expression of 22462 genes is measured. The data set additionally includes labels for each cell denoting their cell type and tumor of origin.

4.2 Evaluation Metrics

To quantitatively evaluate the quality of embeddings two different measures are used in this work. By quality of the embeddings we are interested in how well the local structure of the data is preserved. The choice of the two metrics used in this work follows those used in the original work on parametric t-SNE [52]. A short description for each is given below.

4.2.1 k-Nearest Neighbor Classifier

Given a labeled data set a k-nearest neighbor (k-NN) classifier can be used to roughly assess how well different classes are separated in the output space of an embedding. A k-NN classifier works simply by returning the majority label of a given point's k-nearest neighbors in the training data set. For example in the 1-NN classifier case, the decision boundary can be geometrically visualized as a Voronoi diagram as in Figure 4.1.

Throughout the experiments, the performance of nearest neighbor classifiers in the output space of embeddings will be evaluated. More precisely, the accuracy of a 1-nearest neighbor classifier with the Euclidean metric for determining neighbor distance is used.

For parametric methods, the classifier can be trained on the embedding of training data points and separately tested on the training set. This enables evaluating the ability of the embedding function to generalize to unseen data. On the other hand, for non-parametric methods, the final embedded data points need to be split into train and test sets for the 1-NN classifier.

4.2.2 Trustworthiness

Trustworthiness, introduced by Venna *et al.* [88], is a measure of the degree to which local structure is preserved in the output space relative to the input space after dimensionality reduction is performed. Trustworthiness has been used to compare multiple dimensionality reduction methods for the purposes of visualization of high dimensional data sets such as gene expression data [40] and gene interaction graphs [89].

Computation of trustworthiness is done by comparing how the rank order for each point's k nearest neighbors in the output space match that of the input space. Formally, for a chosen number k of neighbors it is given by the following

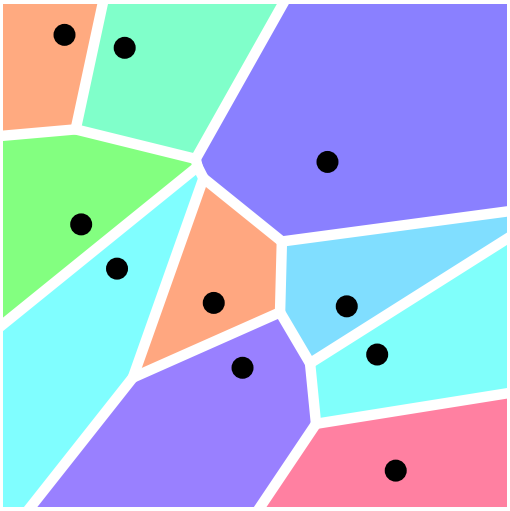

Figure 4.1. *Voronoi diagram illustrating the decision boundaries of 10 different classes of a 1-NN classifier. Black dots represent the training data points, and the different colors correspond to the label of the data point for that region. Any point to be classified would receive the label corresponding to the location in the tesselation it lies on. The distance metric used in this example is the Euclidean distance.*

$$T(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i^k} (r(i,j) - k),$$

where n is the total number of data points under consideration, $\mathcal{N}_i^k$ is the set of k nearest neighbors in the output space of data point i, and $r(i,j)$ is a function giving the rank of node j in the nearest neighbor ordering of node i. The possible values for trustworthiness range from 0 to 1, indicating complete loss of local structure and perfect preservation of local structure, respectively.

4.3 Network Structure and Parameters

To parameterize the embedding $f(\hat{\mathbf{x}})$ a feedforward neural network with layer dimensions $d - 500 - 500 - 2000 - 2$ were used, where d is the input dimensionality and rectifier linear unit (ReLU) activations are applied to the hidden layer outputs to induce nonlinearity to the network. The hidden layer dimensions were chosen to match those of the final combined network used in [52], although we have substituted the sigmoid activations with ReLU activations. Furthermore, we do not perform stacked restricted Boltzmann machine pretraining on the hidden layer weights as in [52]. However we do benchmark the performance of our network against the original parametric t-SNE implementation. Since we are not considering RBM pretraining and are instead relying on the property of ReLUs reducing the vanishing gradient problem [23] and better initialization [22]. Running the original implementation shows little benefit in performing this costly initialization. We could instead reuse the trained encoder weights of the VAE as a starting point for the optimization of our embedding network.

For optimization of the neural network [52] use nonlinear conjugate gradient descent, whereas Adam [42] is used as the optimizer in the results presented here. Updates with Adam are considerably faster to compute than with conjugate gradient descent. Throughout all experiments standard parameters for Adam were chosen, i.e. a learning rate of 0.001 and the exponential decay rates of the 1st and 2nd moments were set to 0.9 and 0.999 respectively. The more recent optimizer AMSGrad [66] was considered, but due to little evidence of its benefit over Adam when used in non-synthetic optimization problems its use was left for future work.

We applied batch normalization [36] to the hidden layers of the VAE networks, while omitting batch normalization from the embedding network. This is to validate the training procedure on as simple an embedding network as possible. We note that applying batch normalization to the embedding network improves the results marginally.

To further emphasize the universality of our approach we restrict ourselves to use a simple VAE structure, without complicating the model architecture with more recently proposed advances, such as normalizing flows.[1] Throughout all experiments, unless otherwise specified, the VAE architecture was fixed to $d - 256 - 128 - 32 - \mu, \log\sigma^2 - 32 - 128 - 256 - d$, where μ and σ parameterize a normal distribution acting as the posterior $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ and the final layer output was used to parameterize a Bernoulli distribution for MNIST as well as Fashion-MNIST, and a Gaussian distribution with a fixed standard deviation of 0.1 for cytometry data. In all experiments, each VAE was trained with a batch size of 1000 for 10000 iterations and had a latent dimensionality of 3, unless otherwise specified.

For MNIST and Fashion-MNIST perplexities were set to 30, which is in line with the value used by [52], whereas for the mass cytometry and scRNA-seq data sets a perplexity of 10 was chosen. It should be noted that a more correct way to choose perplexity would take into account the batch size, as perplexity controls the number of neighbors each point is considered to have. For example training VPTSNE on MNIST with batch size 200 and perplexity 10 gives better results than with perplexity 30. However, with PTSNE the results are actually worse as the training fails to converge even with small learning rates. An example embedding of MNIST with low perplexity and small batch size has been included in Appendix A, which achieves better trustworthiness (0.935) and 1-NN (0.812) than the 600 batch size, 30 perplexity runs. Why PTSNE fails to converge and VPTSNE does not is an important question to explore in future work, as this demonstrates a clear advantage of the method being proposed.

Degrees of freedom of the t-Distribution for the PTSNE objective was set to 1 throughout all experiments, as all embeddings performed were two-dimensional.

[1] In preliminary results VAE models that achieve a lower ELBO do in fact contribute to improved embeddings.

4.4 Results

4.4.1 Learning

In this section we compare the effect of training on VAE reconstructions to training on the original MNIST data set. To quantitatively evaluate the quality of the embeddings produced we employed the trustworthiness metric [88] on the MNIST test set. Additionally, we compared the 1-NN classification errors by fitting the classifiers on the produced embeddings of the training set and finding the mean accuracies of the classifiers on the test set.

In figure 4.2 we have plotted the 1-NN scores and trustworthiness of the embeddings obtained after each iteration of training for two different batch sizes. The runs for each batch size were repeated 20 times, plotting the means and 95% confidence intervals of the means. Training with VAE reconstructions shows a clear improvement both qualitatively and quantitatively over training on the original data when small training batches are used to approximate the t-SNE loss gradient. Higher trustworthiness, as well as 1-NN scores are obtained consistently and convergence is reached in fewer iterations. Moreover, the results are in favor of training on reconstructions by exhibiting more stable results during training.

Qualitatively the embeddings trained on the original data remain noisier than the embeddings trained on the reconstructed data points. This can be seen in figure 4.3, where the separation of true clusters in the embedding is less evident, with several classes overlapping and a large number of outlier points for each class. We further compare the qualitative differences of the embeddings produced with the Fashion-MNIST dataset. In figure 4.4 similar deficiencies in the embedding can be noted as in the embedding comparisons for the MNIST data set. In particular, the visible clusters of classes are less distinct, as well as the global layout of the classes is considerably worse, e.g. the cluster of footwear related images has been pulled closer to the cluster of upper body garments and the images of bags have been split into two seemingly unrelated clusters.

4.4.2 Comparisons

We now compare the proposed method to other parametric dimensionality reduction methods, namely PCA and VAE. In addition we also consider two non-parametric dimensionality reduction methods, t-SNE and the recently proposed method UMAP [60], which had been demonstrated to be able to produce embeddings comparable to t-SNE while being more efficient to optimize in practice.

For the comparisons we will be using the Cytometry A data set. As evaluation criteria, similarly to the previous section, both trustworthiness and 1-NN classification accuracy are used. In addition to comparing the quality of the embeddings, the runtime performance of the different methods are studied. Our goal is to demonstrate that our method produces competitive embeddings as well as has superior scalability.

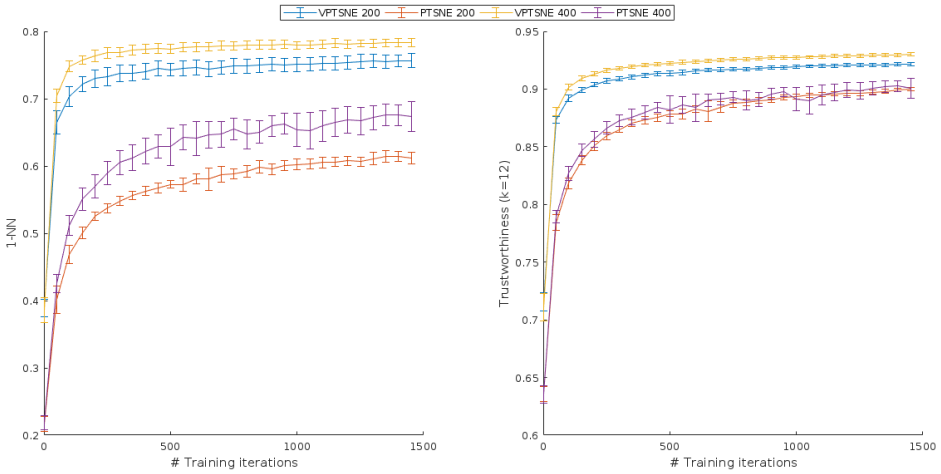

Figure 4.2. *Plots of 1-NN and trustworthiness scores obtained after a given number of iterations for different batch sizes. The means and 95% confidence intervals of the means have been plotted from 20 repeated runs for each parameter setting.*

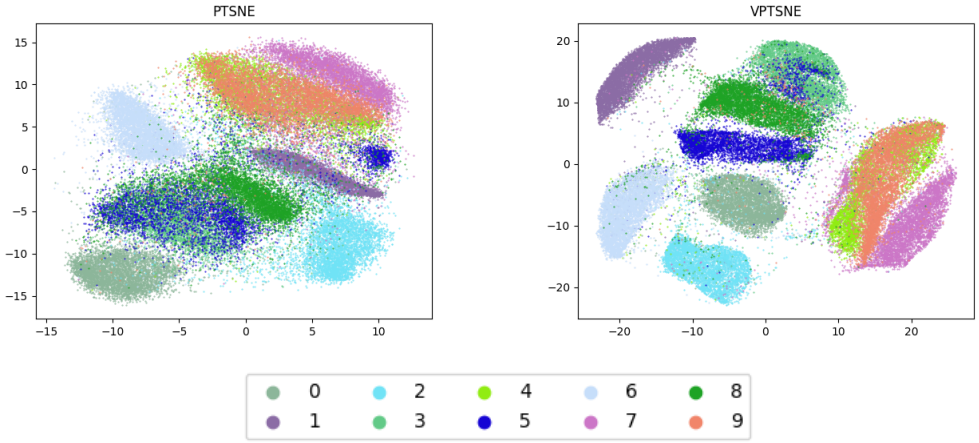

Figure 4.3. *Embeddings of the MNIST data set trained with batch size 400.*

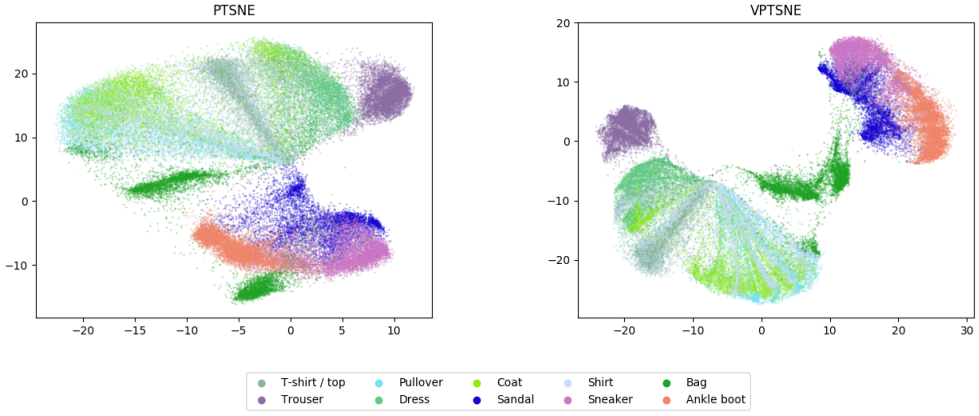

Figure 4.4. *Embeddings of the Fashion-MNIST data set trained with batch size 400.*

Embedding Quality

The quantitative results for all methods under comparison are available in table 4.1 and the corresponding scatter plots of the 2-D embeddings are presented in figure 4.5. Although VAE produces quantitatively better results for the chosen metrics, the spatial layout of clusters in the resulting embedding have less distinct separation and the global structure of the embedding is constrained by the chosen prior $p(\mathbf{z})$. From a data visualization standpoint these factors make the latent space embeddings qualitatively worse than what the corresponding quantitative metrics would suggest. Unsurprisingly, PCA performs the worst on both quantitative and qualitative results. Our method however is able to reach results on par with those of the chosen non-parametric methods.

It can also be observed that t-SNE, with the given perplexity, preserves global structure much more poorly. This can be observed by comparing with the PCA plot, which has three clearly distinct, spatially separated clusters of clusters that are not present in the t-SNE plot.

The effect of choosing a suitable perplexity w.r.t. batch size mentioned in Section 4.3 was noted in these experiments. With large perplexity the clusters in the t-SNE embedding begin to separate more distinctly like with VPTSNE and UMAP, but here the same perplexity is used for both VPTSNE and t-SNE optimization. As perplexity is increased t-SNE optimization becomes considerably slower, for example going from perplexity 10 to 100 increased the total computation time twofold.

Scalability

A major practical challenge with nonlinear dimensionality reduction methods are their scalability to large data sets. Typically, to capture the spatial structure of data a distance metric is employed to compute the pairwise distances for the entire data set, after which a projection of the points to a lower dimensional space, equipped with a corresponding metric, is sought that retains the computed distances as accurately as possible. As the number of data points increases, the inherent quadratic complexity of methods relying on such pairwise distance computations quickly renders such methods intractable.

In practice, various approximations and data structures are employed to speed up dimensionality reduction methods. With t-SNE a widely used performance enhancement, independently investigated by Yang *et al.* [95] and van der Maaten [53], is to use the Barnes-Hut algorithm [5] to approximate the Q_{ij} matrix by averaging the influence of distant points belonging to the same node in the underlying space partitioning data structure. This approximation nevertheless fails to provide significant speedup in the general case depending on the chosen tradeoff between accuracy and speed, as well as the distribution of the points being optimized at each step. Furthermore, its use is limited to obtaining 2 or 3 dimensional embeddings due to its use of the quadtree and octree data structures for each respective dimensionality. On the other hand, optimization of the parametric embedding

Table 4.1. *Comparison between different dimensionality reduction methods.*

Algorithm	Trustworthiness ($k = 12$)	1-NN	time (ms)
VPTSNE	0.9753	0.9250	60766
PTSNE	0.9628	0.8890	8729
UMAP	0.9688	0.9297	74716
VAE	0.9762	0.9247	52037
t-SNE	0.9879	0.9536	1192812
PCA	0.8557	0.4886	96

can be performed with stochastic gradient descent. As we have shown, even with small batch sizes we are able to produce embeddings of competitive quality. The Barnes-Hut approximation can also be used to further speed up the computation of the loss and its gradient, but for small batch sizes the computational overhead of constructing the required data structure can outweigh the cost of directly computing all pairwise influences within the batch. Direct computation has the additional benefit that it is not limited to output dimensionalities for which a space partitioning data structure can be efficiently built.

As previously mentioned, one of the strengths of UMAP compared to t-SNE is the efficiency of its optimization. The key to its efficient optimization is that its gradient can be approximated stochastically via probabilistic edge sampling following the procedure introduced in [83]. As both VPTSNE and UMAP take advantage of stochastic gradient descent, the scalability of their optimization is equal up to an implementation dependent constant factor. VPTSNE however maintains the advantage over UMAP that natural out-of-sample extension is made possible through the learned embedding function.

4.4.3 Robustness to Sparse Data

As discussed in Section 3.4 the probabilistic model is advantageous when only few data points are available. To show this effect in practice we trained mappings with and without the use of our method on randomly chosen subsets of the MNIST training data set, effectively simulating sparsity of available training data. Evaluation of the mappings was carried out on the full MNIST test set as in previous experiments. A line plot of the obtained scores corresponding to the chosen size of the subsets can be found in figure 4.6. Significant differences in the obtained scores are noted in favor of our method. From these results we can observe that both the trustworthiness and 1-NN scores remain higher even with extremely small data sets, indicating that the use of our method is able to provide better generalization even when few training data points are available.

4.4.4 Robustness to Noisy Data

We aim to back up the hypothesis of Section 3.4 by running experiments on artificially corrupted MNIST data sets. As the corruption process in our experiments we consider *masking noise* as in [91], where a predetermined fraction of randomly chosen elements of

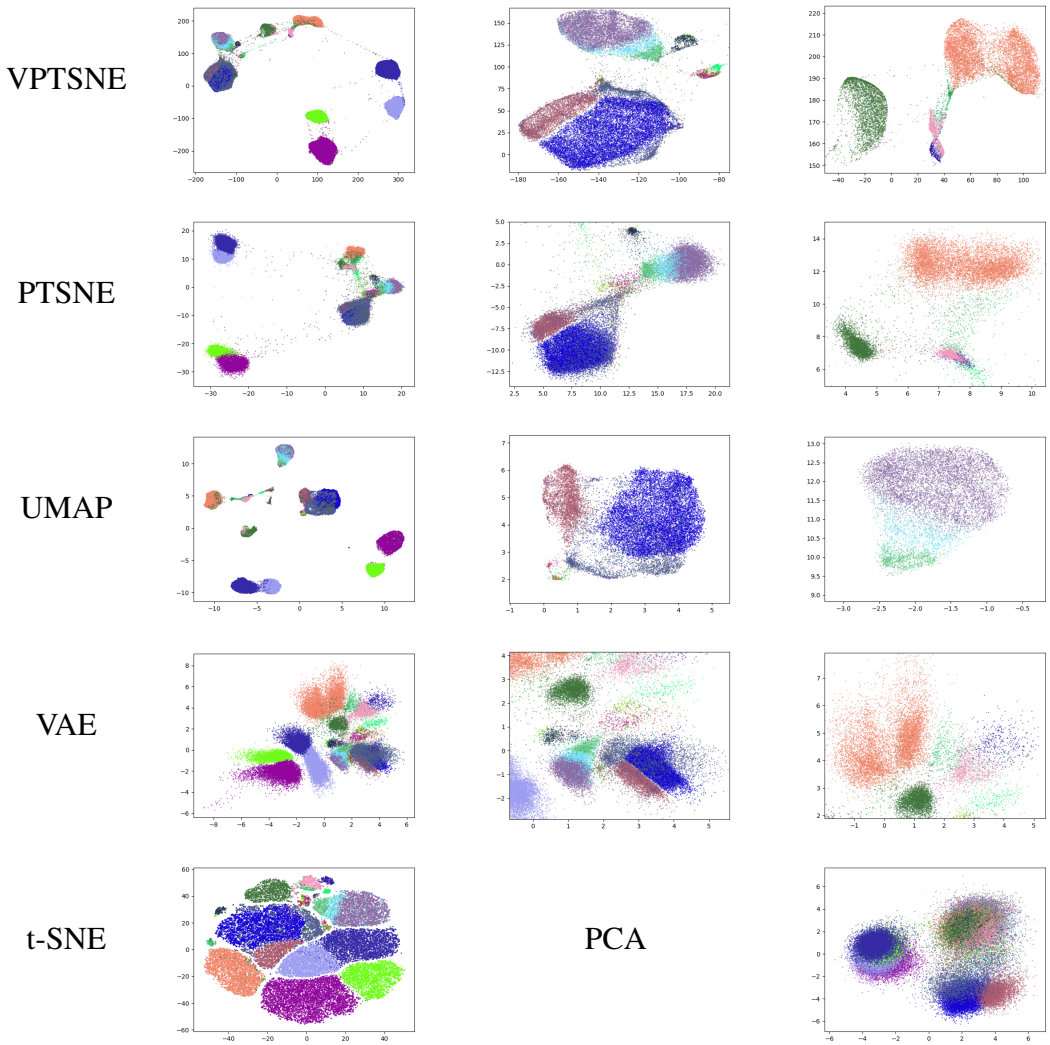

Figure 4.5. *Cytometry A data set embedded with the methods being compared in Section 4.4.2.*

a data point are set to 0. We test robustness by applying masking noise to 10%, 20%, 30% and 40% fractions of the training and test sets. The results of our experiments presented in Figure 4.7 show a clear advantage to using our method when dealing with noisy data as both evaluation metrics can be observed to decrease considerably for PTSNE as the training data is progressively corrupted, whereas the performance of VPTSNE suffers less throughout the experiments.

As an example of real-world data corrupted in this fashion, in scRNA-seq so-called *dropout events* [27], which cause incomplete sequence data to be captured due to technical and biological noise, can be viewed as a form of masking noise. The results presented here would indicate that the proposed method is potentially a good fit for exploratory analysis of scRNA-seq data, even without attempting to explicitly correct for the present technical noise, as in e.g. [65] where factor analysis is extended to additionally model dropouts.

Even with the standard VAE structure, the obtained results are encouraging. As future work

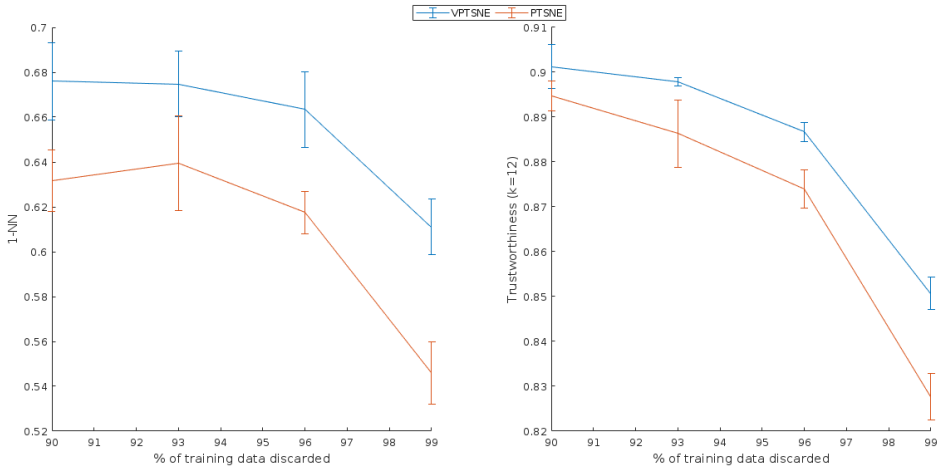

Figure 4.6. *1-NN and trustworthiness scores obtained for PTSNE and VPTSNE trained on small subsets of the MNIST data set. Error bars indicate the 95% confidence intervals of the mean for each sparsity level, obtained from 20 repeated runs.*

it would be interesting to test whether making use of denoising variational autoencoders [35], which are an extension of denoising autoencoders [91] that are directly trained to correct corrupted data. Additionally of interest is how our method fares when other corruption processes are present.

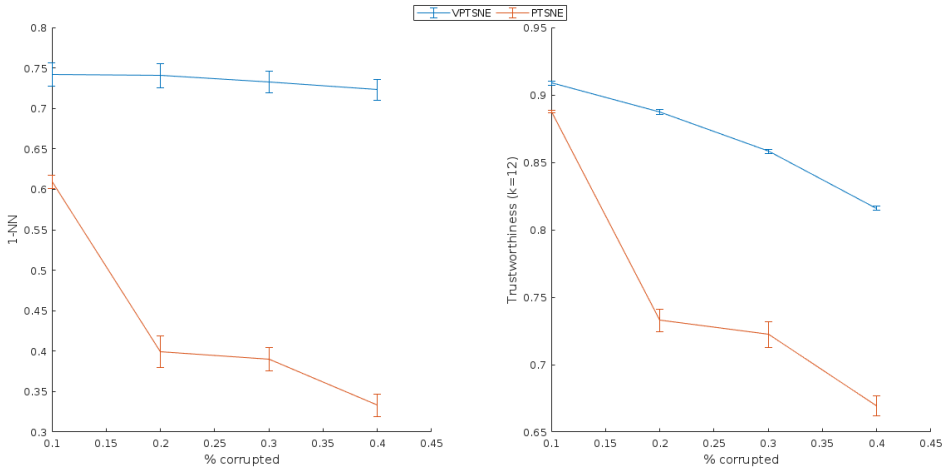

Figure 4.7. *1-NN and trustworthiness scores obtained for PTSNE and VPTSNE trained on MNIST data with different levels of masking noise applied. Error bars indicate the 95% confidence intervals of the mean for each corruption level, obtained from 20 repeated runs.*

4.4.5 Obtaining Reconstructions from Hidden Layers

To show the efficacy of this modification to the learning procedure we train an embedding using the latent code directly. As we choose the dimensionality of the latent code to be more than two orders of magnitude smaller than that of the original data's it is possible for us to use batches of greater size to train the embedding network, without incurring

additional computation cost over training with smaller batches on data of the original dimensionality. The latent dimensionality of the VAE was chosen to be 7 and the batch size for training the embedding was set to 5000, as in the original work on parametric t-SNE [52]. The choice of 7 as the VAE latent dimensionality was deliberately made so that it would correspond to the largest number which is two orders of magnitude smaller than the original dimensionality. We have additionally included the embedding of data points obtained with the original implementation of parametric t-SNE for comparison in figure 4.8. Qualitatively and quantitatively an overall better embedding is achieved with less computation using our method.

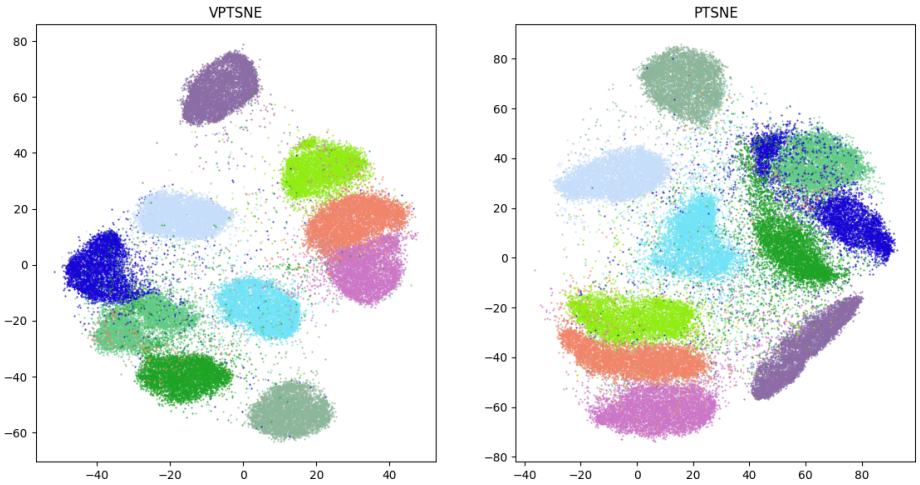

Figure 4.8. *Embedding produced by training on a 7 dimensional latent representation of the MNIST data set contrasted with the result of the original PTSNE implementation. VPTSNE trustworthiness: 0.927, 1-NN: 0.910. PTSNE trustworthiness: 0.926, 1-NN 0.887.*

4.5 Application to Single-Cell Data Analysis

4.5.1 Inference with the Generative Model

As an example of inference with the generative model discussed in 3.3 we use the Cytometry B data set to train our combined model with two patients' data held out. The protein expression profile of Patient A's cells closely match that of other patients' in the training set, whereas Patient B's sample is from advanced cancer making it an outlier in relation to the data available during training. As these cells feature a distinct phenotype associated with a poor response to treatment, they might confer resistance to chemotherapy.

To distinguish between outlier and inlier points in the obtained mapping we set a hard threshold of $\log p(\mathbf{x}) < -150$ to separate between the two. In figure 4.9 embeddings with both patient specific labeling and labeling obtained by thresholding are shown side by side. The high tumor purity sample of Patient B has almost completely been mapped to a

small region corresponding to protein expression levels of cancer cells, whereas Patient A's mapping contains several distinct clusters of various cell types. Further, the labels obtained through thresholding indicate that the majority of cells in Patient B's sample are indeed considered as outliers by the model, in addition to a handful of visually poorly mapped points.

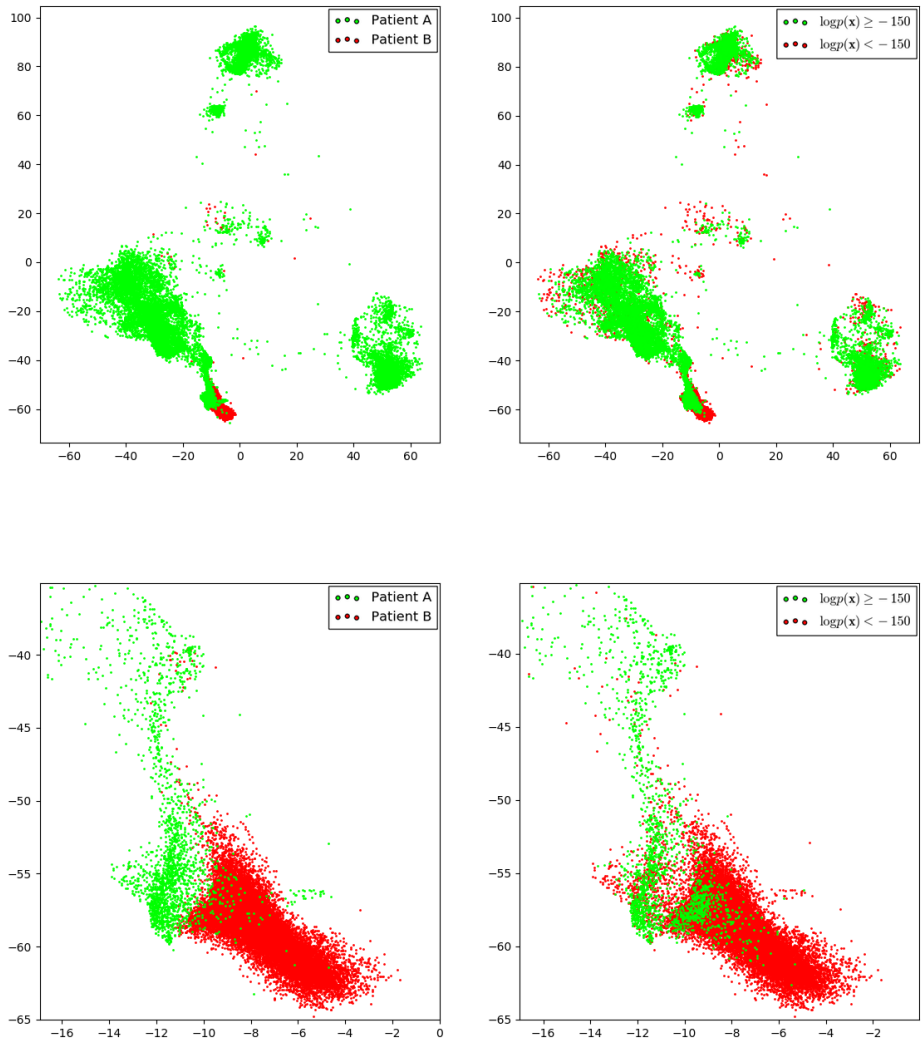

Figure 4.9. *Detecting outliers in cytometry data.*

4.5.2 Application to scRNA-seq Data

Noting the results presented in the previous sections, we finally aim to demonstrate the applicability of our method to a real-world data set that is both sparse and very high dimensional, as well as contains corruption due to technical variation. For this purpose the scRNA-seq data set of [85] was chosen.

Due to the very high dimensionality in comparison to the other data sets, more capacity to

the VAE network was added. The layer dimensions for the VAE architecture were chosen to be $d - 1000 - 1000 - 500 - \mu, \log \sigma^2 - 500 - 1000 - 1000 - d$ and 50 was chosen as the latent dimensionality. To train the embedding we used the method of obtaining the reconstructed data points directly from the latent code.

In Figure 4.10 we have visualized the data of [85] using our method and the results of the analysis carried out in the study.

From the results we can see that different tumor samples and cell types cluster well in the embedding. The separation of malignant cells from different patients to separate clusters reveals the heterogeneity of melanoma present in the different tumor samples.

For each cell in the data set so-called MITF- and AXL-program scores were computed from the relative expression levels of their corresponding gene sets, as in [85]. These scores aim to provide insight into the potential drug resistance mechanisms of malignant cells [21, 46]. In our embedding a clear gradient from high to low MITF- and AXL-program scores can be seen within the cluster of tumor cell clusters, further indicating that our method is able to highlight biologically meaningful variations in the data.

Cell cycle analysis also reveals the presence of subclusters of cycling cells in our embedding. Cycling cells being cells which are undergoing cell division. The color gradient in the projection for cycling cells corresponds to the strength of the signal indicating cell cycling. Below a given threshold value for this signal, cells were classified as non-cycling.

With such a high dimensional data set, the effect of the curse of dimensionality can be seen on methods based on distances, leading them to produce inferior embeddings. For example, applying UMAP directly to the high dimensional data, a couple of the most obvious shortcomings of the embedding is its failure to distinguish between different cell types as well as being unable to properly separate tumor cell clusters originating from different samples. A figure of the UMAP embedding is included in Appendix B for comparison.

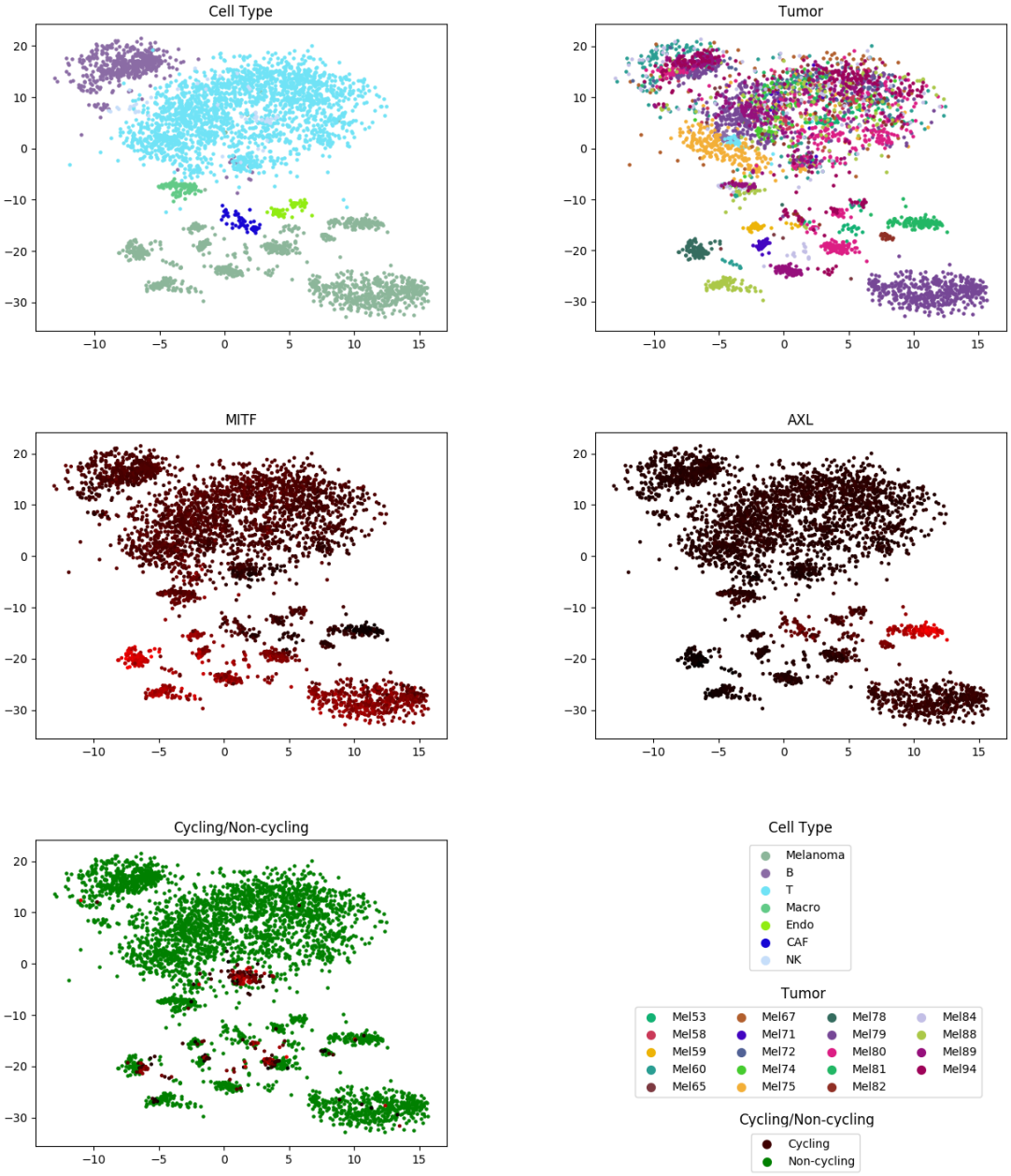

Figure 4.10. *Visualization of analysis results on data from [85] using VPTSNE.*

5. CONCLUSIONS

By combining two dimensionality reduction methods: a stochastic variant of autoencoders and a neural network parameterized variant of t-SNE, we demonstrated a method for parametric dimensionality reduction exhibiting various advantageous properties in comparison to either method applied by itself. [Note: .. recap results ..]

In addition, two applications of the method to single-cell data analysis were demonstrated in this work. The method was applied to both moderately high dimensional proteomic data and extremely high dimensional transcriptomic data. [Note: .. recap more results ..]

[Note: .. Why development of new methods is important [to better analyze ever more complex data, gathered with continuously developing technology] ..]

[Note: .. Future work ..]

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, Tensorflow: a system for large-scale machine learning., in: OSDI, 2016, Vol. 16, pp. 265–283.

[2] C.C. Aggarwal, A. Hinneburg, D.A. Keim, On the surprising behavior of distance metrics in high dimensional space, in: International conference on database theory, Springer, 2001, pp. 420–434.

[3] Amir El-ad David, Davis Kara L, Tadmor Michelle D, Simonds Erin F, Levine Jacob H, Bendall Sean C, Shenfeld Daniel K, Krishnaswamy Smita, Nolan Garry P, Pe'er Dana, viSNE enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia, Nature biotechnology, Vol. 31, Iss. 6, may, 2013, p. 545–552.

[4] P. Baldi, K. Hornik, Neural networks and principal component analysis: Learning from examples without local minima, Neural networks, Vol. 2, Iss. 1, 1989, pp. 53–58.

[5] Barnes Josh, Hut Piet, A hierarchical O(N log N) force-calculation algorithm, Nature, Vol. 324, dec, 1986, p. 446.

[6] D.M. Blei, A. Kucukelbir, J.D. McAuliffe, Variational inference: A review for statisticians, Journal of the American Statistical Association, Vol. 112, Iss. 518, 2017, pp. 859–877.

[7] C.J.C. Burges, Dimension reduction: A guided tour, Foundations and Trends® in Machine Learning, Vol. 2, Iss. 4, 2010, pp. 275–365.

[8] C.P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, A. Lerchner, Understanding disentangling in β-VAE, ArXiv e-prints, Apr. 2018.

[9] E.J. Candès, X. Li, Y. Ma, J. Wright, Robust principal component analysis?, Journal of the ACM (JACM), Vol. 58, Iss. 3, 2011, p. 11.

[10] V. Chandrasekaran, S. Sanghavi, P.A. Parrilo, A.S. Willsky, Rank-sparsity incoherence for matrix decomposition, SIAM Journal on Optimization, Vol. 21, Iss. 2, 2011, pp. 572–596.

[11] T.Q. Chen, X. Li, R. Grosse, D. Duvenaud, Isolating sources of disentanglement in variational autoencoders, arXiv preprint arXiv:1802.04942, 2018.

[12] Clarke Robert, Ressom Habtom W., Wang Antai, Xuan Jianhua, Liu Minetta C., Gehan Edmund A., Wang Yue, The properties of high-dimensional data spaces: implications for exploring gene and protein expression data, Nature Reviews Cancer, Vol. 8, jan, 2008, p. 37.

[13] J. Cook, I. Sutskever, A. Mnih, G. Hinton, Visualizing similarity data with a mixture of maps, in: Artificial Intelligence and Statistics, 2007, pp. 67–74.

[14] B. Dai, Y. Wang, J. Aston, G. Hua, D. Wipf, Connections with robust pca and the role of emergent sparsity in variational autoencoder models, Journal of Machine Learning Research, Vol. 19, Iss. 41, 2018, pp. 1–42.

[15] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, Ieee, 2009, pp. 248–255.

[16] J.V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, R.A. Saurous, Tensorflow distributions, arXiv preprint arXiv:1711.10604, 2017.

[17] W. Dong, C. Moses, K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures, in: Proceedings of the 20th international conference on World wide web, ACM, 2011, pp. 577–586.

[18] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research, Vol. 12, Iss. Jul, 2011, pp. 2121–2159.

[19] S.E. Fahlman, G.E. Hinton, T.J. Sejnowski, Massively parallel architectures for ai: Netl, thistle, and boltzmann machines, in: National Conference on Artificial Intelligence, AAAI, 1983.

[20] K.P. F.R.S., LIII. On lines and planes of closest fit to systems of points in space, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Vol. 2, Iss. 11, 1901, pp. 559–572.

[21] Garraway Levi A., Widlund Hans R., Rubin Mark A., Getz Gad, Berger Aaron J., Ramaswamy Sridhar, Beroukhim Rameen, Milner Danny A., Granter Scott R., Du Jinyan, Lee Charles, Wagner Stephan N., Li Cheng, Golub Todd R., Rimm David L., Meyerson Matthew L., Fisher David E., Sellers William R., Integrative genomic analyses identify MITF as a lineage survival oncogene amplified in malignant melanoma, Nature, Vol. 436, jul, 2005, p. 117.

[22] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Teh, Y.W., Titterington, M. (eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Chia La-

guna Resort, Sardinia, Italy, 13–15 May, 2010, Proceedings of Machine Learning Research 9, PMLR, pp. 249–256.

[23] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011), Apr. 2011.

[24] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Nets, in: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 2672–2680. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[26] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[27] S.C. Hicks, F.W. Townes, M. Teng, R.A. Irizarry, Missing data and technical variability in single-cell rna-sequencing experiments, Biostatistics, Vol. 19, Iss. 4, 2018, pp. 562–578.

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

[29] G.E. Hinton, Learning multiple layers of representation, Trends in cognitive sciences, Vol. 11, Iss. 10, 2007, pp. 428–434.

[30] G.E. Hinton, S. Osindero, Y.W. Teh, A fast learning algorithm for deep belief nets, Neural computation, Vol. 18, Iss. 7, 2006, pp. 1527–1554.

[31] G.E. Hinton, S.T. Roweis, Stochastic neighbor embedding, in: Advances in neural information processing systems, 2003, pp. 857–864.

[32] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, science, Vol. 313, Iss. 5786, 2006, pp. 504–507.

[33] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural networks, Vol. 4, Iss. 2, 1991, pp. 251–257.

[34] H. Hotelling, Relations between two sets of variates, Biometrika, Vol. 28, Iss. 3/4, 1936, pp. 321–377.

[35] D.J. Im, S. Ahn, R. Memisevic, Y. Bengio, Denoising criterion for variational auto-encoding framework, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA., 2017, pp. 2059–2065.

[36] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, CoRR, Vol. abs/1502.03167, 2015.

[37] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax.

[38] W.B. Johnson, J. Lindenstrauss, Extensions of lipschitz mappings into a hilbert space, Contemporary mathematics, Vol. 26, Iss. 189-206, 1984, p. 1.

[39] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, In-datacenter performance analysis of a tensor processing unit, in: Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on, IEEE, 2017, pp. 1–12.

[40] S. Kaski, J. Nikkilä, M. Oja, J. Venna, P. Törönen, E. Castrén, Trustworthiness and metrics in visualizing similarity of gene expression, BMC Bioinformatics, Vol. 4, Iss. 1, Oct, 2003, p. 48.

[41] N.S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P.T.P. Tang, On large-batch training for deep learning: Generalization gap and sharp minima, in: International Conference on Learning Representations, 2017.

[42] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR, Vol. abs/1412.6980, 2014.

[43] D.P. Kingma, S. Mohamed, D.J. Rezende, M. Welling, Semi-supervised learning with deep generative models, in: Advances in Neural Information Processing Systems, 2014, pp. 3581–3589.

[44] D.P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, M. Welling, Improved variational inference with inverse autoregressive flow, in: Advances in Neural Information Processing Systems, 2016, pp. 4743–4751.

[45] D.P. Kingma, M. Welling, Auto-encoding variational bayes., CoRR, Vol. abs/1312.6114, 2013.

[46] D.J. Konieczkowski, C.M. Johannessen, O. Abudayyeh, J.W. Kim, Z.A. Cooper, A. Piris, D.T. Frederick, M. Barzily-Rokni, R. Straussman, R. Haq, D.E. Fisher, J.P. Mesirov, W.C. Hahn, K.T. Flaherty, J.A. Wargo, P. Tamayo, L.A. Garraway, A melanoma cell state distinction influences sensitivity to mapk pathway inhibitors, Cancer Discovery, Vol. 4, Iss. 7, 2014, pp. 816–827.

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

[48] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, D.M. Blei, Automatic differentiation variational inference, The Journal of Machine Learning Research, Vol. 18, Iss. 1, 2017, pp. 430–474.

[49] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, nature, Vol. 521, Iss. 7553, 2015, p. 436.

[50] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Vol. 86, Dec. 1998, pp. 2278 – 2324.

[51] Levine Jacob H., Simonds Erin F., Bendall Sean C., Davis Kara L., Amir El-ad D., Tadmor Michelle D., Litvin Oren, Fienberg Harris G., Jager Astraea, Zunder Eli R., Finck Rachel, Gedman Amanda L., Radtke Ina, Downing James R., Pe'er Dana, Nolan Garry P., Data-Driven Phenotypic Dissection of AML Reveals Progenitor-like Cells that Correlate with Prognosis, Cell, Vol. 162, Iss. 1, doi: 10.1016/j.cell.2015.05.047, feb, 2018, pp. 184–197.

[52] L. Maaten, Learning a parametric embedding by preserving local structure, in: Dyk, D. van, Welling, M. (eds.), Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr, 2009, Proceedings of Machine Learning Research 5, PMLR, pp. 384–391.

[53] L. van der Maaten, Accelerating t-SNE using Tree-Based Algorithms, Journal of Machine Learning Research, Vol. 15, 2014, pp. 3221–3245.

[54] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, Journal of Machine Learning Research, Vol. 9, 2008, pp. 2579–2605.

[55] L. Van der Maaten, G. Hinton, Visualizing non-metric similarities in multiple maps, Machine learning, Vol. 87, Iss. 1, 2012, pp. 33–55.

[56] L. van der Maaten, E. Postma, H. van den Herik, Dimensionality reduction: A comparative review.

[57] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, Adversarial autoencoders, in: International Conference on Learning Representations, 2016.

[58] S. Markidis, S.W. Der Chien, E. Laure, I.B. Peng, J.S. Vetter, Nvidia tensor core programmability, performance & precision, arXiv preprint arXiv:1803.04014, 2018.

[59] J. Martens, R. Grosse, Optimizing neural networks with kronecker-factored approximate curvature, in: International conference on machine learning, 2015, pp. 2408–2417.

[60] L. McInnes, J. Healy, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, ArXiv e-prints, Feb. 2018.

[61] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski *et al.*, Human-level control through deep reinforcement learning, Nature, Vol. 518, Iss. 7540, 2015, p. 529.

[62] H. Narayanan, S. Mitter, Sample complexity of testing the manifold hypothesis, in: Advances in Neural Information Processing Systems, 2010, pp. 1786–1794.

[63] R.M. Neal, Annealed importance sampling, Statistics and computing, Vol. 11, Iss. 2, 2001, pp. 125–139.

[64] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with cuda, Queue, Vol. 6, Iss. 2, Mar. 2008, pp. 40–53.

[65] E. Pierson, C. Yau, Zifa: Dimensionality reduction for zero-inflated single-cell gene expression analysis, Genome Biology, Vol. 16, Iss. 1, Nov, 2015, p. 241.

[66] S.J. Reddi, S. Kale, S. Kumar, On the convergence of adam and beyond, in: International Conference on Learning Representations, 2018.

[67] D. Rezende, S. Mohamed, Variational inference with normalizing flows, in: Bach, F., Blei, D. (eds.), Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 07–09 Jul, 2015, Proceedings of Machine Learning Research 37, PMLR, pp. 1530–1538.

[68] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, in: Xing, E.P., Jebara, T. (eds.), Proceedings of the 31st International Conference on Machine Learning, Bejing, China, 22–24 Jun, 2014, Proceedings of Machine Learning Research 32, PMLR, pp. 1278–1286.

[69] V. Rokhlin, Rapid solution of integral equations of classical potential theory, Journal of computational physics, Vol. 60, Iss. 2, 1985, pp. 187–207.

[70] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, California Univ San Diego La Jolla Inst for Cognitive Science, Techn. rep., 1985.

[71] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, nature, Vol. 323, Iss. 6088, 1986, p. 533.

[72] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision (IJCV), Vol. 115, Iss. 3, 2015, pp. 211–252.

[73] R. Salakhutdinov, G. Hinton, Deep boltzmann machines, in: Dyk, D. van, Welling, M. (eds.), Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr, 2009, Proceedings of Machine Learning Research 5, PMLR, pp. 448–455.

[74] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift), ArXiv e-prints, May 2018.

[75] S. Shalev-Shwartz, S. Ben-David, Understanding machine learning: From theory to algorithms, Cambridge university press, 2014.

[76] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, Mastering the game of go with deep neural networks and tree search, nature, Vol. 529, Iss. 7587, 2016, p. 484.

[77] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, Science, Vol. 362, Iss. 6419, 2018, pp. 1140–1144.

[78] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, Mastering the game of go without human knowledge, nature, Vol. 550, Iss. 7676, 2017, p. 354.

[79] S.L. Smith, P.J. Kindermans, Q.V. Le, Don't decay the learning rate, increase the batch size, in: International Conference on Learning Representations, 2018.

[80] M.H. Spitzer, G.P. Nolan, Mass Cytometry: Single Cells, Many Features, Cell, Vol. 165, Iss. 4, 2016, pp. 780–791.

[81] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research, Vol. 15, Iss. 1, 2014, pp. 1929–1958.

[82] Stegle Oliver, Teichmann Sarah A., Marioni John C., Computational and analytical challenges in single-cell transcriptomics, Nature Reviews Genetics, Vol. 16, jan, 2015, p. 133.

[83] J. Tang, J. Liu, M. Zhang, Q. Mei, Visualizing large-scale and high-dimensional data, in: Proceedings of the 25th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2016, pp. 287–297.

[84] T. Tieleman, G. Hinton, Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning, 2012.

[85] I. Tirosh, B. Izar, S.M. Prakadan, M.H. Wadsworth, D. Treacy, J.J. Trombetta, A. Rotem, C. Rodman, C. Lian, G. Murphy, M. Fallahi-Sichani, K. Dutton-Regester, J.R. Lin, O. Cohen, P. Shah, D. Lu, A.S. Genshaft, T.K. Hughes, C.G.K. Ziegler, S.W. Kazer, A. Gaillard, K.E. Kolb, A.C. Villani, C.M. Johannessen, A.Y. Andreev, E.M. Van Allen, M. Bertagnolli, P.K. Sorger, R.J. Sullivan, K.T. Flaherty, D.T. Frederick, J. Jané-Valbuena, C.H. Yoon, O. Rozenblatt-Rosen, A.K. Shalek, A. Regev, L.A. Garraway, Dissecting the multicellular ecosystem of metastatic melanoma by single-cell rna-seq, Science, Vol. 352, Iss. 6282, 2016, pp. 189–196.

[86] J.M. Tomczak, M. Welling, Improving variational auto-encoders using householder flow, CoRR, Vol. abs/1611.09630, 2016.

[87] L. Van Der Maaten, K. Weinberger, Stochastic triplet embedding, in: Machine Learning for Signal Processing (MLSP), 2012 IEEE International Workshop on, IEEE, 2012, pp. 1–6.

[88] J. Venna, S. Kaski, Neighborhood preservation in nonlinear projection methods: An experimental study, in: Dorffner, G., Bischof, H., Hornik, K. (eds.), Artificial Neural Networks — ICANN 2001, Berlin, Heidelberg, 2001, Springer Berlin Heidelberg, pp. 485–491.

[89] J. Venna, S. Kaski, Visualizing gene interaction graphs with local multidimensional scaling, in: ESANN, 2006.

[90] R. Vershynin, High-dimensional probability: An introduction with applications in data science, Vol. 47, Cambridge University Press, 2018.

[91] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, Journal of machine learning research, Vol. 11, Iss. Dec, 2010, pp. 3371–3408.

[92] A.C. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht, The marginal value of adaptive gradient methods in machine learning, in: Advances in Neural Information Processing Systems, 2017, pp. 4148–4158.

[93] Y. Wu, Y. Burda, R. Salakhutdinov, R.B. Grosse, On the quantitative analysis of decoder-based generative models, CoRR, Vol. abs/1611.04273, 2016.

[94] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[95] Z. Yang, J. Peltonen, S. Kaski, Scalable optimization of neighbor embedding for visualization, in: International Conference on Machine Learning, 2013, pp. 127–135.

[96] S. Zhao, J. Song, S. Ermon, Infovae: Information maximizing variational autoencoders, arXiv preprint arXiv:1706.02262, 2017.

APPENDIX A: SMALL BATCH SIZE, LOW PERPLEXITY EMBEDDING

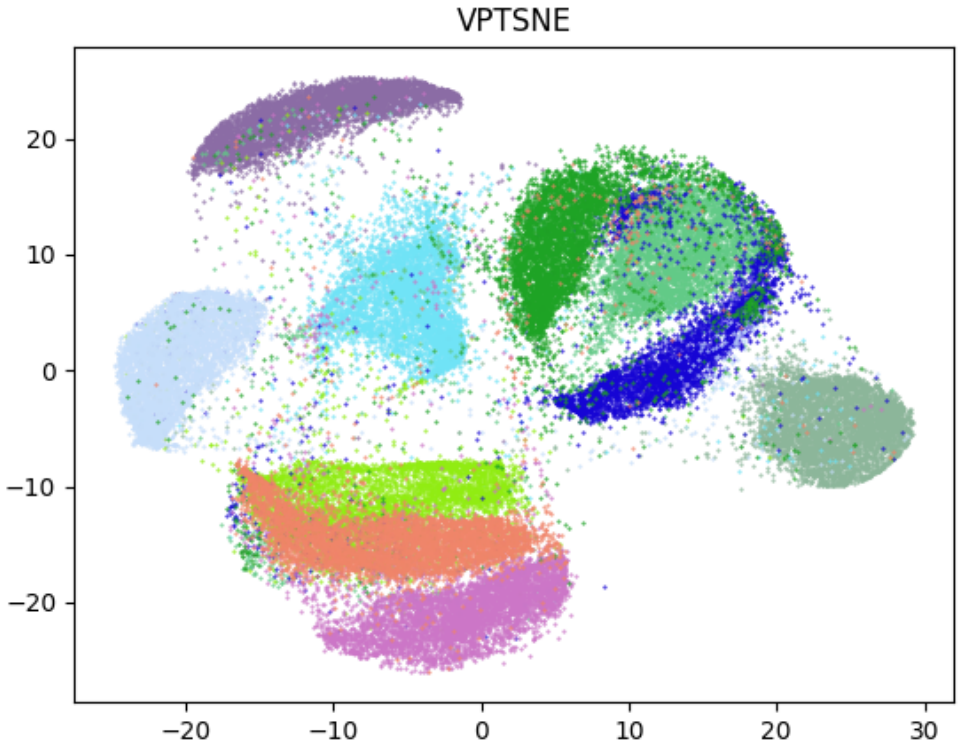

Figure A.1. *Effect of the choice of perplexity w.r.t. batch size. MNIST VPTSNE embedding trained with batch size 200 and perplexity 10. Trustworthiness: 0.935, 1-NN: 0.812.*

APPENDIX B: UMAP EMBEDDING OF SCRNA-SEQ DATA

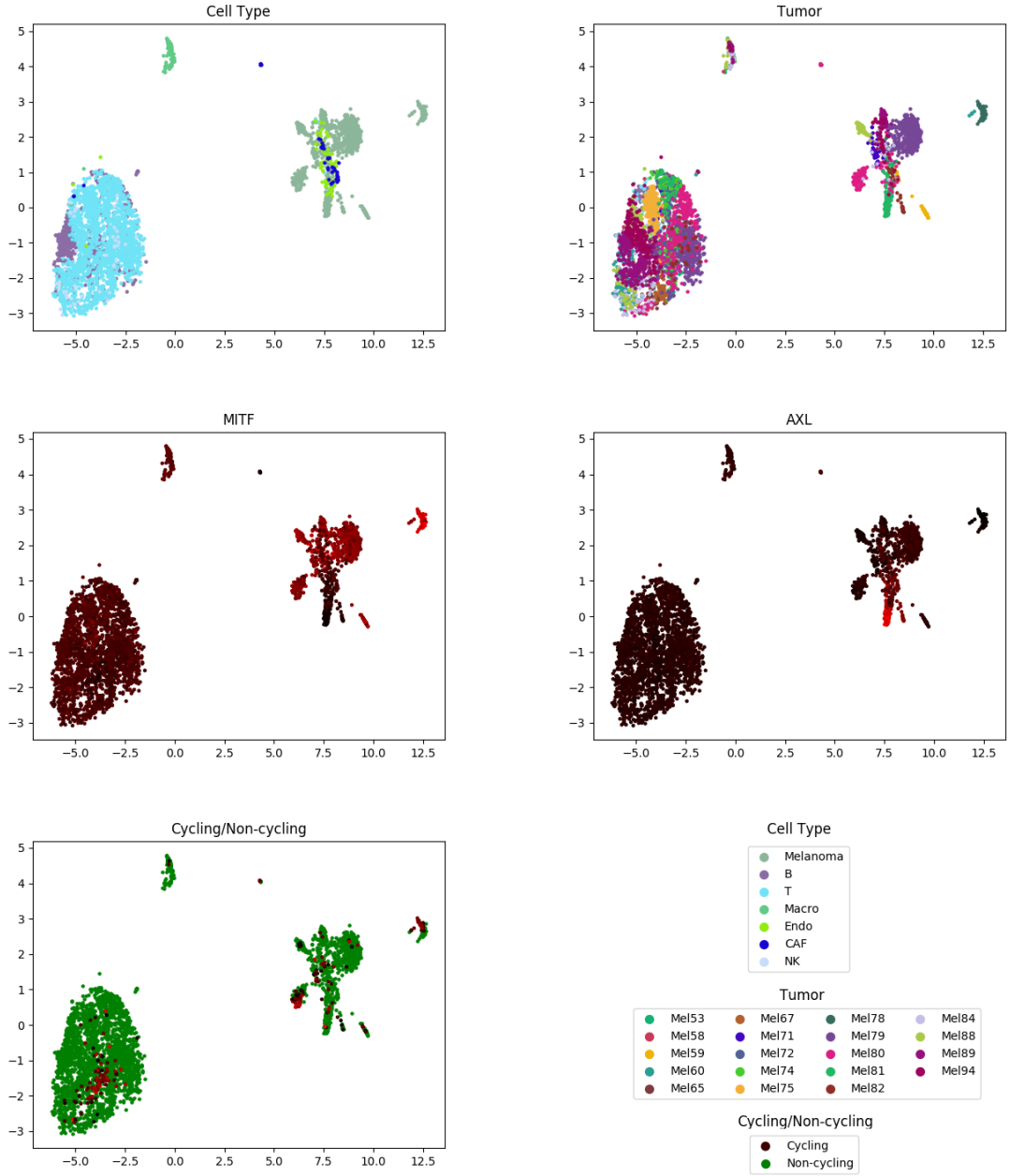

Figure B.1. *Visualization of analysis results on data from [85] using UMAP.*