List of topics to cover

With section titles and brief explantions.

Yiftach Kolb

Berlin, October 12, 2022

Abstract

punkt. punkt.

Declaration

punkt. punkt.

Acknowledgement

punkt.[3] punkt. bip/bop/boop **XxZzYyWw**$\mathbb{Z}$ so-so–so—so

List of Tables

List of Figures

Contents

Chapter 1

Introduction

punkt. *punkt*, *punkt*.

Chapter 2

Notations and definitions, preliminary concepts

2.1 Basic notations

Throughout this paper (modulo typing errors) we use capital bold math Latin or Greek letters $(\boldsymbol{X}, \boldsymbol{\Sigma})$ to represent matrices. To stress that we talk about matrices rather than vectors we show product $(\times)$ in the dimension, i.e $\boldsymbol{X} \in \mathbb{R}^{m \times n}$. Although technically the matrix–space is the tensor product $\mathbb{R}^m \otimes \mathbb{R}^n$.

Bold small math letters $(\boldsymbol{x})$ represent usually row vectors, but in cases where it makes sense may also represent matrices such as a batch of several vectors (each row is a different data point). In few occasions it makes sense to let it represent both a matrix and a vector, for example, $\boldsymbol{\sigma}$ may represent both the covariance matrix and the variance vector of a diagonal Gaussian distribution. Non-bold math letters $(x, \sigma, \dots)$ may represent scalar or vectors in some cases and hopefully it is clear from the context or explicitly stated.

Since we are only dealing with real matrices the transpose and the conjugation operators are the same $(A^T = A^*)$ but over $\mathbb{C}$ conjugation is usually the "natural" operation and we use it to indicate that some property is still valid over $\mathbb{C}$ with conjugation.

Sometimes matrices are given in row/column/block notations inside brackets where the elements are concatenated in a way that makes positional sense. For example both $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}|\mathbf{y})$ represent a matrix with 2 **columns**.

As mentioned usually just $\mathbf{x}$ means a column vector and $\mathbf{x}^T$ means a row vector but sometimes in matrix notation $\mathbf{x}$ represents a row when it makes sense. We use **curly** brackets to indicate the **row** representations of a matrix. For example $\{\mathbf{x}, \mathbf{y}\}$ represents a matrix whose **rows** are $\mathbf{x}$ and $\mathbf{y}$ (as row vectors), which alternatively could be represented as $(\mathbf{x}, \mathbf{y})^T$.

$(\mathbf{X}, \mathbf{Y})$ represents concatenation of two matrices which implicitly means they have the same number of rows.

Zero–blocks are indicated with 0 or are simply left as voids. For example $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \mathbf{0} & \boldsymbol{D} \end{pmatrix}$ represents block notation of an upper–triangular matrix.

Definition 2.1. Let $\mathbf{X} = \{\mathbf{x}_1, \dots \mathbf{x}_m\} \in \mathbb{R}^{m \times n}$ be a matrix in **row** notation. Then its *squared Frobenius norm* is

$$\|X\|_F^2 \triangleq \operatorname{trace}(\mathbf{X}\mathbf{X}^*) = \sum_{i=1}^{m} \|\mathbf{x}_i\|_2^2 = \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^2 \tag{2.1}$$

2.2 The data

we assume that the input data unless otherwise stated is real and 2-dimensional. Its rows represent *samples*, for example—cells in the case of scRNAseq dataset, while its columns represent *variables*, for example—genes in scRNAseq dataset.

There could possibly be additional data matrices with information about class or conditions. We use *one-hot encoding* to represent such information.

Definition 2.2. A *data matrix* is simply a real–valued matrix $\boldsymbol{X} \in \mathbb{R}^{N \times n}$ which represent a set of N n-dimensional data points. The N rows are also called *observations* and the n columns are *variables.*

Definition 2.3. A *class matrix*, or also a *condition matrix* $\boldsymbol{C} \in \mathbb{R}^{\mathbb{N} \times c}$ is simply a real matrix which represents one-hot encoding of c classes or conditions over N samples. For example if sample i has class j, then $(\forall k \in 1, \dots, c)\boldsymbol{C}[i,k] = \delta_{jk}$.

We say that that $\boldsymbol{C}$ is a *class probability matrix* or a *relaxed class matrix* (same with condition) if instead of being one-hot it is a distribution matrix, namely each row is non-negative and sums up to 1.

Usually if the input data includes class/condition information, it comes as a class matrix (pure one-hot) but the output (the prediction) is naturally probabilistic and hence is relaxed.

2.3 Linear algebra preliminary: SVD and PCA

In the following state some facts and bring without proof what are the singular value decomposition and the principle components of a matrix. For a full proof see [7].

Let $\boldsymbol{X} \in \mathbb{R}^{N \times n}$ be a real-valued matrix representing N samples of some n-dimensional data points and let $r = \operatorname{rank}(\boldsymbol{X}) \leq \min(n, N)$.

$\mathbf{X}\mathbf{X}^*$ and $\mathbf{X}^*\mathbf{X}$ are both symmetric and positive semi-definite. Their eigenvalues are non-negative, and they both have the same positive eigenvalues, exactly r such, which we mark $s_1^2 \geq s_2^2 \geq \dots s_r^2 > 0$. The values $s_1 \dots s_r$ are called the *singular values* of $\boldsymbol{X}$.

Let $\boldsymbol{S} = \begin{pmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_r \end{pmatrix} \in \mathbb{R}^{r \times r}$

Let $\boldsymbol{U} = (\boldsymbol{u}_1 | \dots | \boldsymbol{u}_N) \in \mathbb{R}^{N \times N}$ be the (column) right eigenvectors of $\mathbf{X}\mathbf{X}^*$ sorted by their eigenvalues. Then $\boldsymbol{U} = (\boldsymbol{U}_r, \boldsymbol{U}_k)$ where $\boldsymbol{U}_r = (\boldsymbol{u}_1 | \dots | \boldsymbol{u}_r) \in \mathbb{R}^{N \times r}$ are the first

r eigenvectors corresponding to the non-zero eigenvalues, and $\boldsymbol{U}_k$ are the eigenvectors corresponding to the $N-r$ 0-eigenvalues. Similarly let $\boldsymbol{V} = (\boldsymbol{V}_r, \boldsymbol{V}_k) \in \mathbb{R}^{n\times n}$ be the (column) right eigenvectors of $\mathbf{X}^*\mathbf{X}$, sorted by the eigenvalues, where $\boldsymbol{V}_r = (\boldsymbol{v}_1|\dots|\boldsymbol{v}_r) \in \mathbb{R}^{n\times r}$ are the firs r eigenvalues and $\boldsymbol{V}_k$ are the $n-r$ null-eigenvectors.

The critical observations is that $\boldsymbol{V}_r = \boldsymbol{X}^*\boldsymbol{U}_r S^{-1}$ and then $\boldsymbol{U}_r^*\mathbf{X}\boldsymbol{V}_r = S$.

The *singular value decomposition (SVD)* of $\mathbf{X}$ is

$$\mathbf{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^* \tag{2.2}$$

where $\boldsymbol{D} = \begin{pmatrix} \boldsymbol{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{N\times n}$ is diagonal.

$\boldsymbol{V}_r$ are called the *(right) principle components* of $\mathbf{X}$. Note that $\boldsymbol{V}_r^*\boldsymbol{V}_r = \boldsymbol{I}_r$ and that $\mathbf{X} = \mathbf{X}\boldsymbol{V}_r\boldsymbol{V}_r^* = (\mathbf{X}\boldsymbol{V}_r)\boldsymbol{V}_r^T$. If one looks at the second expression, it means that the each row of $\mathbf{X}$ is spanned by the orthogonal basis $\boldsymbol{V}_r^T$ (because the other vectors of $\boldsymbol{V}$ are in $\ker(\mathbf{X})$.

More generally For every $l \le r$, let $\boldsymbol{V}_l \in \mathbb{R}^{N\times l}$ be the first l components, Then $\mathbf{X}\boldsymbol{V}_l\boldsymbol{V}_l^T$ is as close as we can get to $\mathbf{X}$ within an l-dimensional subspace of R^n, and $\boldsymbol{V_l}$ minimizes

$$\boldsymbol{V}_l = \mathrm{argmin}_{\mathbf{W}}\{\|\mathbf{X} - \mathbf{X}\boldsymbol{W}\boldsymbol{W}^T\|_F^2 \quad : \quad \boldsymbol{W} \in \mathbb{R}^{n\times l}, \boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}_l\} \tag{2.3}$$

Where $\|\cdot\|_F^2$ is simply the sum of squares of the matrix' entries.

If we consider the more general minimization problems:

$$\min_{\boldsymbol{E},\boldsymbol{D}}\{\|\mathbf{X} - \mathbf{X}\boldsymbol{E}\boldsymbol{D}\|_F^2 \quad : \quad \boldsymbol{E}, \boldsymbol{D^T} \in \mathbb{R}^{n\times l},\} \tag{2.4}$$

$$\min_{\mathbf{W}}\{\|\mathbf{X} - \mathbf{X}\boldsymbol{W}\boldsymbol{W}^\dagger\|_F^2 \quad : \quad \boldsymbol{W} \in \mathbb{R}^{n\times l},\} \tag{2.5}$$

It can be shown [5] that the last two problems 2.4, 2.5 are equivalent and that for any solution E, D it must hold that $\boldsymbol{D} = \boldsymbol{E}^\dagger$. ($\boldsymbol{D}$ is the Moore–Penrose generalized inverse of $\boldsymbol{E}$). Moreover, $\boldsymbol{V}_l$ still minimizes the general problem 2.4 and for every solution $\boldsymbol{W}$, it must hold that $\mathrm{span}\{\boldsymbol{W}\} = \mathrm{span}\{\boldsymbol{V}_l\}$ (but it isn't necessarily an orthogonal matrix).

2.4 Neural networks

Definition 2.4. A *feed forward neural network* is simply a parameterized differentiable map $\phi_\omega : \mathbb{R}^n \to \mathbb{R}^m$. $\phi_\omega(x)$ is differentiable both in its input variable x as well as in its parameters ω, which are also called its *weights.*

For example each affine function has the form $f : x \mapsto a\cdot x + b$. a and b are its trainable parameters (weights). The input is treated as fixed (the data we are trying to explain).

Normally ϕ is a sequence of compositions of more simple functions. We call such more basic unit in a composition sequence a *layer.* Each layer is itself a composition, with exactly a single affine map, followed by 0 or more dimension preserving functions such as normalization functions or activation functions. An *activation function* is a real values non-linear function which is applied element-wise over the input later. For example the sigmoid function and the ReLU (rectified linear unit) are well-known and often used activation functions.

Definition 2.5. Usually together with a neural network comes an associated differentiable function which is the *loss function* $\mathcal{L} : \mathbb{R}^m \to \mathbb{R}$.

Typically the loss function is additive on the dimension, meaning it has the form $\mathcal{L}(\boldsymbol{x}) = \sum_{i=1}^{m} \psi(x_i)$

An example for such a loss function is the square error: $\mathbf{x} \mapsto \|\mathbf{x}\|_2^2$

Definition 2.6. Let $\boldsymbol{X} \in \mathbb{R}^{N \times n}$ be a data matrix. A *batch* $\boldsymbol{x} \in \mathbb{R}^{b \times n}$ is any subset of b rows of $\boldsymbol{X}$ (Note that in this case $\mathbf{x}$ represents a matrix).

Batch $\boldsymbol{x} = \{\mathbf{x}_1, \dots \mathbf{x}_b\} \in \mathbb{R}^{b \times n}$ (row notation) represents a subset of b samples out of the total of N samples in the dataset. The operations of ϕ and $\mathcal{L}$ *extend* to batches in natural ways— collect for ϕ and we average for $\mathcal{L}$, namely:

Definition 2.7. Let ϕ be a neural network as defined in 2.4 and let $\mathcal{L}$ its associated loss function as defined in 2.5—over vectors. Let $\mathbf{x} = \{\mathbf{x}_1, \dots, \mathbf{x}_b\} \in \mathbb{R}^{b \times n}$ be a b-batch (in row notation). Then ϕ and $\mathcal{L}$ *extended* over batches are:

$$\phi(\boldsymbol{x}) \triangleq \{\phi(\boldsymbol{x}_i)\}_{i=1}^{m} \in \mathbb{R}^{b \times m} \tag{2.6}$$

$$\mathcal{L}(\mathbf{x}) \triangleq \frac{1}{b} \sum_{i=1}^{b} \mathcal{L}(\boldsymbol{x}_i) \in \mathbb{R} \tag{2.7}$$

If $\mathcal{L}$ is the square error function $\|\cdot\|_2^2$ on vectors, then its extension to batches is $\frac{1}{b}\|\cdot\|_F^2$. The reason why we sum and don't average over the dimensions will be cleared later when we get into variational inference.

Training the neural network ϕ_w means finding the weights that minimize the loss function applied on the training set X, in other words minimizing $\min_w(\mathcal{L}(\phi_w(X))$.

Definition 2.8. Let ϕ_ω be a neural network and $\mathcal{L}$ its associated loss function. And let the data matrix $\mathbf{X}$ be our *training set.* Then *Training* of ϕ_ω with respect to $\mathcal{L}, \mathbf{X}$ means algorithmically approximating the minimization problem:

$$\min_{\omega} \mathcal{L}(\phi_\omega(\mathbf{X})) \tag{2.8}$$

During a *training step* the network is applied on a batch x. Then the loss function is applied on the output and a gradient (with relation to the weights) is taken. This gradient is used for the weight update rule, which varies depending on the specific training algorithm. Typical training algorithms are SGD (stochastic gradient decent) and Adam, which is the one used throughout this work.

We only need to define the network, the loss function and the specific training algorithm. The rest (derivation, weight update etc.) is taken care for us by the backend of the software (Pytorch [6]) and can be regarded as a black box.

2.5 Autoencoders

Definition 2.9. An *Autoencoder* (AE) is a neural network $\phi = D \circ E$ with a "bottleneck" layer and which approximates the identity function on the training input.

We call E which projects into the bottleneck, the *encoder*, and D which expands back into the input dimensions, the *decoder*.

2.5.1 Relation between PCA and AE

For **centered** data, meaning every variable (column of $\boldsymbol{X}$) has 0 sample mean, the first $k \leq \text{rank}(\boldsymbol{X})$ principle components $\boldsymbol{P}$ are the solution for equation 2.3; Whereas a **linear** autoencoder solves equation 2.5. As mentioned, it must hold that $E = D^\dagger$ (the encoder must be the Moore-Penrose inverse of the decoder).

A linear autoencoder (an AE where ϕ is linear) is therefore almost equivalent to PCA [5], in that in the optimum, a bottleneck space of dimension k is spanned by the first k principle components of the input $\boldsymbol{X}$. In general, an AE can be seen a PCA-like, but non-linear method for dimensionality reduction.

Chapter 3

Variance inference and variational autoencoders

3.1 Variational Inference

Here we briefly explain the idea behind variational inference and introduce the ELBO which is the loss function we'll use throughout this text. For more details see Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning.* Vol. 4. 4. Springer, 2006.

We treat the data matrix as a set of independent observations (its rows) $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ which we try to explain by a probabilistic model. We assume that the $\boldsymbol{x}_i$'s are i.i.d with some distribution function $p(\boldsymbol{x})$ and therefore for the entire dataset it holds that $p(\boldsymbol{X}) = \prod p(\boldsymbol{x}_i)$.

Definition 3.1. Let $\boldsymbol{X} \in \mathbb{R}^{Nn}$ be a data matrix and let $\{\boldsymbol{x}_i\}_1^n$ be its rows, which we assume to be i.i.d with some (unknown) distribution $p(\boldsymbol{x})$. Then $\log p(\boldsymbol{X}) = \sum_1^N p(\boldsymbol{x}_i)$ is called the *log evidence* of our data.

The r.vs $\boldsymbol{X}$ are high dimensional however we have some reason to believe that behind the scenes there are some hidden (latent), smaller dimensional, r.vs $\boldsymbol{Z} = \{\boldsymbol{z}_1 \dots \boldsymbol{z}_N\}$ that generate the observations $\boldsymbol{X}$. In other words we think that $\boldsymbol{X}$ is conditioned on $\boldsymbol{Z}$ and we can speak of the joint distribution $p(\boldsymbol{X}, \boldsymbol{Z}) = p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z})$. Because we assume i.i.d for both $\boldsymbol{X}$ and $\boldsymbol{Z}$ all the distributions factor over the individual samples multiplicatively, e.g. $p(\boldsymbol{X}|\boldsymbol{Z}) = \prod p(\boldsymbol{x}_i|\boldsymbol{z}_i)$.

Suppose that we have a fully Bayesian model. In this case there are no parameters because the parameters are themselves stochastic variables with some suitable priors. We can therefore pack all the latent variables and stochastic parameters into one latent "meta variable" $\boldsymbol{Z} = (\boldsymbol{z}_1, \boldsymbol{z}_2, \dots)$, where each $\boldsymbol{z}_i$ is some multidimensional r.v and possibly composed of several simpler r.vs (for example a categorical and a normal r.vs). We similarly pack all the observed variables into one meta variable $\boldsymbol{X}$. Together we have a distribution $p(\boldsymbol{X}, \boldsymbol{Z})$ and the working assumption is that it is easy to factorize $p(\boldsymbol{X}, \boldsymbol{Z}) = p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z})$, however $p(\boldsymbol{Z}|\boldsymbol{X})$ is intractable and $p(\boldsymbol{X})$ is unknown.

We are being Bayesian here so we consider $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots)$ to be a constant a set of observations and we want to best explain $p(\boldsymbol{X})$ by finding as high as possible lower bound for it (or rather to $\log p(\boldsymbol{X})$, the *log evidence*). A second goal is to approximate the intractable $p(\boldsymbol{Z}|\boldsymbol{X})$ by some simpler distribution $q(\boldsymbol{Z})$ taken from some family of distributions.

Definition 3.2. Let $\boldsymbol{x}, \boldsymbol{z}$ be random variables with joint distribution $p(\boldsymbol{x}, \boldsymbol{z})$ and let $q(\boldsymbol{z})$ be any distribution. The *evidence lower bound (ELBO)* with respect to p, q is:

$$\mathcal{L}(q,p) = \triangleq \int \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})} dq(\boldsymbol{z}) \tag{3.1}$$

The following equation shows that the *ELBO* is a lower bound for the *log evidence.* (using Jensen's inequality)

$$\begin{aligned}\log p(\boldsymbol{X}) &= \log \int p(\boldsymbol{X},\boldsymbol{Z}) d\boldsymbol{Z} = \log \int \frac{p(\boldsymbol{X},\boldsymbol{Z})}{q(\boldsymbol{Z})} q(\boldsymbol{Z}) d\boldsymbol{Z} \\ &= \log \int \frac{p(\boldsymbol{X},\boldsymbol{Z})}{q(\boldsymbol{Z})} dq(\boldsymbol{Z}) \geq \int \log \frac{p(\boldsymbol{X},\boldsymbol{Z})}{q(\boldsymbol{Z})} dq(\boldsymbol{Z}) \triangleq \mathcal{L}(q,p)\end{aligned} \tag{3.2}$$

In equation 3.2 we found a lower bound $\mathcal{L}(q,p)$ for the log evidence $\log p(\boldsymbol{X})$, the *ELBO.* Whatever distribution q we put in ELBO will not be greater than the real log evidence so we are looking for the q which **maximizes** it.

Now we show that maximizing the ELBO actually obtains the log evidence and it is equivalent to minimizing $KL(q(\boldsymbol{Z})\|p(\boldsymbol{Z}|\boldsymbol{X})$:

$$\begin{aligned}\mathcal{L}(q,p) &\triangleq \int \log \frac{p(\boldsymbol{X},\boldsymbol{Z})}{q(\boldsymbol{Z})} dq(\boldsymbol{Z}) = \int \log \frac{p(\boldsymbol{Z}|\boldsymbol{X})p(\boldsymbol{X})}{q(\boldsymbol{Z})} dq(\boldsymbol{Z}) \\ &= \int \log p(\boldsymbol{X}) dq(\boldsymbol{Z}) - \int \log \frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z}|\boldsymbol{X})} dq(\boldsymbol{Z}) = \log p(\boldsymbol{X}) - KL(q(\boldsymbol{Z})\|p(\boldsymbol{Z}|\boldsymbol{X})\end{aligned} \tag{3.3}$$

We can rewrite equation 3.3 as:

$$\log p(\boldsymbol{X}) = \mathcal{L}(q,p) - KL(q(\boldsymbol{Z})\|p(\boldsymbol{Z}|\boldsymbol{X})) \tag{3.4}$$

Equation 3.4 shows that the ELBO minus the kl-divergence are constant and equal the log evidence. Therefore minimizing the kl-divergence (which is always non-negative) simultaneously maximizes the ELBO and vicer-versa.

3.2 Variational Autoencoder

3.2.1 Adding parameters

Our models will not be fully Bayesian, but rather parametrized. In this case let θ represent the set of parameters for p, and ϕ the parameters for q. Meaning we are dealing with a family of distributions $p_\theta(x, z)$ and another family $q_\phi(z)$.

For any θ and any ϕ, the equations from the previous chapter hold also in the parametrize form, i.e $\log p_\theta(x) = \mathcal{L}(q_\phi) - KL(q_\phi(Z)\|p_\theta(Z|X)$.

We assume that we can only approach the "real" distribution using θ from below $\log p(X) \geq \log p_\theta(X)$. So together with equation 3.2 we have

$$(\forall\theta,\phi) \log p(X) \geq \log p_\theta(X) \geq \mathcal{L}(q_\phi) = \int \frac{p_\theta(X,Z)}{q_\phi(Z)} dq_\phi(Z) \tag{3.5}$$

So from equation 3.4 we again see that by finding the parameters ϕ, θ that maximize the elbo we approach the real log evidence as much as we can within the limits of the parametrized family of distributions we use.

3.2.2 Rearranging the ELBO

Equations 3.2 and 3.3 were defined for any distribution $q(\mathbf{Z})$ and in particular we are allowed to plug in a conditioned distribution $q(\mathbf{Z}|\mathbf{X})$. That implies the existence of $q(\mathbf{Z},\mathbf{X})$ and $q(\mathbf{X})$ but we actually don't care about them. We condition everything on $\mathbf{X}$ but $\mathbf{X}$ is treated as a given constant from a Bayesian view point and we only want to somehow make $q(\mathbf{Z}|\mathbf{X})$ to closely approximate $p(\mathbf{Z}|\mathbf{X})$.

A second thing we need to achieve is to express the ELBO in terms of $p(\mathbf{X}|\mathbf{Z})$ and $q(\mathbf{Z}|\mathbf{X})$ rather than the joint distribution. To that end we need also the prior $p(\mathbf{Z})$.

Definition 3.3. The *conditioned (on* $\mathbf{X}$*) ELBO* is

$$\begin{aligned}\mathcal{L}(q,p) &\triangleq \int \log \frac{p(\mathbf{X},\mathbf{Z})}{q(\mathbf{Z}|\mathbf{X})} dq(\mathbf{Z}|\mathbf{X}) = \int \log \frac{p(\mathbf{X}|\mathbf{Z})p(\mathbf{Z})}{q(\mathbf{Z}|\mathbf{X})} dq(\mathbf{Z}|\mathbf{X}) \\ &= \int \log p(\mathbf{X}|\mathbf{Z}) dq(\mathbf{Z}|\mathbf{X}) - \int \log \frac{q(\mathbf{Z}|\mathbf{X})}{p(\mathbf{Z})} dq(\mathbf{Z}|\mathbf{X}) \\ &= \int \log p(\mathbf{X}|\mathbf{Z}) dq(\mathbf{Z}|\mathbf{X}) - KL(q(\mathbf{Z}|\mathbf{X}) \| p(\mathbf{Z}))\end{aligned} \tag{3.6}$$

So to sum it up, if we want to maximize the log evidence $\log p(\mathbf{X})$ it suffices to maximize II, $\surd$ and equation 3.3 shows that this means finding the balance between making the right term (which we call the reconstruction term) large as possible, and making the KL-term small. The KL term is seen as a regularization term.

3.2.3 Thinking of the data as distribution

We can think of any data point $\mathbf{x}_i \in X$, rather than being a deterministic point, as if is a sample which comes from some distribution p_i. For example if $\mathbf{x}_i$ is a vector of binary data, we can think of it as if it was generated by some Bernoulli distribution. If the data is a in the range$[0,1]$ we can think of every data point as a vector of Bernoulli probabilities rather than a concrete sample. For other type of real data, we think of it as coming from a diagonal Gaussian distribution, where we usually assume the variance is fixed 1 or more generally that all the samples have some common variance σ.

Similarly we can also treat the latent variable as coming from some distribution rather than being deterministic. Actually that is what equation 3.3 is all about.

3.2.4 Using neural networks for the parametrization

In this text we deal with variational autoencoders (VAE). A VAE is a neural network which is used to define and optimize the parameters ϕ and θ which define $p_\theta(\mathbf{X}|\mathbf{Z})$ and $q_\phi(\mathbf{Z}|\mathbf{X})$.

Specifically the encoder part of the network is a non-linear map $f_\theta(\mathbf{Z})$ which is used to define $P_\theta(X|Z)$. For example, we can assume that P_θ is a family of multivariate Gaussians and in this case $f_\theta(Z) = (\mu(Z), \Sigma(Z))$. Meaning the encoder maps Z to the location vector and covariance matrice. The parameter θ in this case are the weights of the encoder neural network. In parametrize the prior $p(Z)$ however in this case its parameter is not a function of X. In practice there is no reason to do this for most VAEs and we choose some simple fixed prior distribution for $p(Z)$.

The decoder network is similarly defined as a non-linear function $g_\phi(X)$ which maps X into the parameters defining the family $q_\phi(Z)$. Here too ϕ represent the weights of the decoder.

Definition 3.4. Let $\{p_\theta\}$ be a family of distributions over $\mathbb{R}^n$ and let $\{q_\phi\}$ be a family of distributions over $\mathbb{R}^m$ and let p be some fixed distribution over $\mathbb{R}^m$. A *variational autoencoder (VAE)* consists of neural networks $f_\omega : \mathbb{R}^n \to \{\theta\}_\theta$ and $g_\omega : \mathbb{R}^m \to \{\phi\}_\phi$ where ω are the combined parameters of f, g and $\{\theta\}$ means the parameter space (all possible values of θ which can define a valid distribution p_ϕ (similarly with ϕ).

If an autoencoder works on deterministic data, with the encoder mapping input $\mathbf{x} \mapsto \mathbf{z}$ and the decoder then maps the latent space $\mathbf{z} \mapsto \hat{x}$ to the reconstruction, a VAE tries to do basically the same thing but non-deterministically. It maps $\mathbf{x}$ into a distribution over $\mathbf{z}$: $\mathbf{x} \mapsto q(\mathbf{z}|\mathbf{x})$ and it maps $\mathbf{z}$ into a distribution over $\mathbf{x}$, $\mathbf{z} \mapsto p(\mathbf{x}|\mathbf{z})$.

3.2.5 Graphical representation

It is both convenient as well as informative to include a graphical description of our probabilitic models by way of plate diagrams.

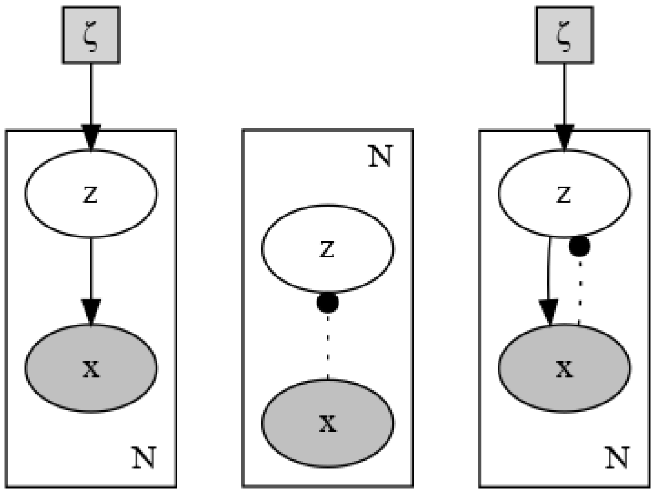
Please note that we drop the ϕ, θ subscript but they are still there in reality.

In a plate diagram nodes represent random variables and arrows represent dependency. Figure 3.1 is a plate diagram of the VAE model with slight adaptation. We use doted arrows to represent the arrows of the inference model, and regular arrows for the generative model. Regular (triangular) arrowhead represents real probabilitic dependency whereas rounded arrows are reminding us that this is not a real probabilitic dependency (recall **??**) which maybe we can call 'parametric dependency'.

Plate represents packing of N i.i.ds since we have N observations $X = (x_i)_1^N$ and correspondingly N latent variables $Z = (z_i)$.

Shaded node represent known values (either observation of prior).

The squared ζ node represent some fixed parameters which describes the prior distribution of $P(z)$. Usually it is not shown in the papers about VAE but we just wanted to remind the reader that it can be parametrize in general.



(a) generative model ($p(x, z)$ (b) inference model ($q(z|x)$) (c) the combined graphical model

Figure 3.1: VAE graphical model

The generative model therefore factors as: $p(x, z) = p(x|z)p(z|\zeta) = p(x|z)p(z)$

The inference model in this case is just $q(z)$ but we might denote is as $q(z|x)$ because it tries to approximate $p(z|x)$.

Note that the graphical model has no assumption about the specific types of distributions involved (Gaussian, Dirichlet or whateve . . .) and that is left for the actual implementation.

In the case of a ”vanilla” VAE, We chose the prior to be diagonal standard Gaussian $p(z) \sim N(; 0, 1)$. And $p(z|x)$ is then chosed as diagonal Gaussian (whose means and variances we find by training the network).

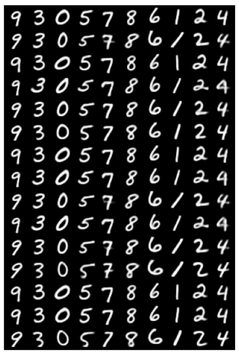
Chapter 4

Gaussian mixture model VAEs

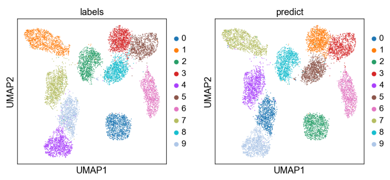
Theoretical background and with some examples from publications and my own tests.

4.0.1 Relation between AE and VAE

4.0.2 Conditional VAE



(a)



(b)

Figure 4.1: a figure

Chapter 5

Experiments and results

5.1 Tests with MNIST and FMNIST

5.2 Tests with scRNAseq Data

some words about (sc)RNAseq and published papers where AE and VAE models have been applied. What we were hoping to achieve and compare with.

Chapter 6

Discussion, some remarks and conclusions

punkt. punkt. punkt.

Bibliography

[1] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning.* Vol. 4. 4. Springer, 2006.

[2] Xifeng Guo et al. “Improved deep embedded clustering with local structure preservation.” In: *Ijcai.* 2017, pp. 1753–1759.

[3] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[4] Mohammad Lotfollahi, F Alexander Wolf, and Fabian J Theis. “Generative modeling and latent space arithmetics predict single-cell perturbation response across cell types, studies and species”. In: *bioRxiv* (2018), p. 478503.

[5] Elad Plaut. “From principal subspaces to principal components with linear autoencoders”. In: *arXiv preprint arXiv:1804.10253* (2018).

[6] Automatic Differentiation In Pytorch. *Pytorch.* 2018.

[7] Denis Serre. “Matrices: Theory & Applications Additional exercises”. In: *L’Ecole Normale Supérieure de Lyon* (2001).