HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTER THESIS

Author:
Yahya Badran

VQ-VAE with Neural Gas and Fuzzy c-Means

2021

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

VQ-VAE with Neural Gas and Fuzzy c-Means

Author:
Yahya Badran

Study Programme:
Applied Mathematics for Networking and Data Science

Seminar Group:
MA18w1-M

First Referee:
Prof. Dr. Thomas Villmann

Second Referee:
Dr. Marika Kaden

Mittweida, May 2021

Acknowledgement

I would like thank Prof.Dr.Thomas Villmann and Dr.Marika Kaden for giving me the opportunity explore this topic and the support they provided through out our studies. I would also like to thank all the teaching staff of this Master program, they gave us a variety of interesting and important topics that made up this unique program.

Not to forget my family and my amazing friends, thank you so much.

Bibliographic Information

Badran, Yahya: VQ-VAE with Neural Gas and Fuzzy c-Means, 35 pages, 4 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences

Master Thesis, 2021

Abstract

VQ-VAE is a successful generative model which can perform lossy compression. It combines deep learning with vector quantization to achieve a discrete compressed representation of the data. We explore using different vector quantization techniques with VQ-VAE, mainly neural gas and fuzzy c-means. Moreover, VQ-VAE consists of a non-differentiable discrete mapping which we will explore and propose changes to the original VQ-VAE loss to fit the alternative vector quantization techniques.

I. Contents

II. List of Figures

III. List of Tables

1 Introduction

1.1 Background

Generative models main goal is to generate new unseen data instances that can pass under some criteria as if they are generated from the the same source of the available data. This can have different applications. For example, performing missing data imputation which is the process of replacing the missing data with plausible alternatives [16]. Another related application is data augmentation which is the process of generating new data that can pass as samples from the original distribution of the data with the purpose of increasing the size of the data set [26].

Besides the goal of synthesising new data, some generative models can build a simpler representation of the data which is often called the latent representation, a famous example is the variational autoencoder (VAE). The latent representation of the data can be used to perform yet other tasks such as classification [5, 29]. However, the latent representation produced by generative models is usually not a good alternative for the non-generative approaches such as the Autoencoder latent representation [29].

In [29], they introduced a generative model called Vector Quantised-Variational AutoEncoder (VQ-VAE) which uses a discrete latent representation. Besides being a competitive generative model, they showed that it can produce useful latent representations. They also showed that it is a viable lossy compression technique [29], where lossy compression is a kind of compression that allows some information loss.

Using a discrete representation can be a natural choice in some domains such as modeling human language [29]. However, discrete mappings are not differentiable and this makes implementing gradient based optimization problematic. Nevertheless, problems which are inherently discrete can employ gradient based optimization successfully. For example, neural gas which is a vector quantization technique.

Vector quantization are unsupervised learning algorithms that can perform clustering. Clustering itself is a discrete mapping. Despite the discrete nature of the problem, different vector quantization models were introduced in the literature with differentiable objective function and thus can be optimized using gradient descent based algorithms

VQ-VAE partly uses vector quantization to optemize its discrete mapping. In [29], they used k-means. In [9], they replaced k-means with self-organizing map (SOM). In [25,30], they used Gaussian mixture inspired approach.

In our text, we will investigate two other vector quantization alternatives with VQ-VAE,

neural gas and fuzzy c-means. We will see that neural gas can provide competitve results. Moreover, to test the different vector quantization methods, we will give a simple generalization of the original VQ-VAE.

1.2 The Structure of the Thesis

In chapter 2, we introduce basic concepts that will be needed in later chapters. Mainly a review of some popular neural network models such as multilayer perceptron, convolutional neural network, and autoencoders.

In chapter 3, we cover different types of generative models that are needed to explain VQ-VAE.

In chapter 4, we review vector quantization and clustering in general with focus on k-means, neural gas, and fuzzy c-means.

In chapter 5, we introduce VQ-VAE and a generalization of the VQ-VAE to implement the different vector quantization methods that we introduced in chapter 4.

Finally, Chapter 6 provides practical results from training some of the described models.

2 Preliminaries

2.1 Artificial Neural Networks

Artificial Neural Networks is a term used to describe mathematical models that are inspired by biological neurons. We will only briefly describe two models: multilayer perceptrons (MLP) and convolutional neural netowks (CNN). We will follow notations from [2]

2.1.1 Multilayer Perceptron (MLP)

An MLP consists of multiple layers. The input data is considered as the first layer and the output of the model is considered as the last layer. This leaves what is called a hidden layer

$$h^l(x^{l-1}) = g(b^l + W^l x^l) \tag{2.1}$$

where l is the layer number, x^l is the output of layer l except for x^0 which is just the input data x. g is a non-linear function called the activation function. W^l is a matrix and b^l is a vector which both are learnable parameter of layer l.

One example of an activation function is the sigmoid

$$\sigma(u) = \frac{1}{1+exp(-u)}, u \in \mathbb{R}$$

which we will need later on in this text.

2.1.2 CNN

CNN take use of an operation used a lot in signal processing and specifically in image processing which is convolution. Convulotion is a mathematical operation between two signals. It's usually used to express a transformation in which an input signal M is transformed into an output signal O by applying a convulution between M and another signal called the kernel K. This operation is represented as

$$O = K * M$$

In our case the signals are just matrices and the operation can be written as

$$O[k,l] = \sum_j \sum_i M[i,j] \cdot K[k-i, l-j] \tag{2.2}$$

Where $O[k,l]$ is the element of row k and column l (which is the same for M and K).

Similar to MLP, CNN can be split into conceptual layers as follows

$$\mathbf{x}_c^\ell = f\left(\sum_{c \in C_{l-1}} \mathbf{x}_c^{\ell-1} * \mathbf{k}_{cj}^\ell + b_j^\ell\right) \tag{2.3}$$

where x_c^ℓ can be a 3d tensors which consists of columns, rows, and channels similar to an RGB image. x_c^ℓ means the channel c of the output of layer ℓ. C_l are the channels available in layer ℓ. While b_j is vector called the bias. K_{cj} is the kernel of layer l that gives output to channel c in the next layer. Both b and K are the learnable parameters.

Note this is not the only way CNN can be implemented. Moreover, notice the flipping of the kernel in equation 2.3 is not important, we will ignore that in the discussion since the kernel is learnable and if flipping is needed we assume it is possible to learn it during training.

2.2 Discrete Distribution

Let z be a discrete random variable that takes N possible values from the set of integers $\{0,1,\ldots,n-1\}$. A model can output a true discrete distribuiton for z by outputing an n-dimensional vector $(p_0, p_1, \ldots, p_{n-1})$ such that the probability of $z = i$ is p_i. The vector elements should satisfy

- $p_i > 0 \quad \forall i \in \{0,1,\ldots,n-1\}$
- $\sum_{i=0}^{n-1} p_i = 1$

One method to enforce these properties is the softmax function. Assume we have the set of real numbers $z_1, z_2, \ldots, z_{n-1} \in \mathbb{R}$ then we can output a discrete distribution as follows

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1,\ldots,K$$

As we will see later, there are other methods to output a discrete distribution.

2.3 Autoencoder

An Autoencoder consists of an encoder, $A_\theta : X \to Z$, and a decoder, $B_\phi : Z \to X$, where Z is called the latent space and X is the data space. Both θ and ϕ are learnable parametrs. The latent representation of the data is its image in the latent space. As it was described in [1] , the goal is to learn an encoder and a decoder that optimize the following loss function

$$L(x) = \mathbb{E}[\Delta(\mathbf{x}, B_\phi \circ A_\theta(\mathbf{x})] \tag{2.4}$$

The expected value is over the true distribution of the data which will be approximated by the empirical mean. Δ is the reconstruction loss which is usually the euclidean distance. The encoder and decoder are usually parameterised by neural networks.

Using such a loss function, forces all the information to pass through the latent space, Z. Thus the encoder has to encode the needed information required by the decoder in order to reconstruct the data. Therefore some representation of the data will be learned and encoded in the latent space, Z. Different purposes for the encoded data give rise to different implementation of the autoencoder.

3 Generative Models

Let $\mathbb{X}$ represent our data set. We assume that $\mathbb{X}$ was sampled from a distribution which we do not know. The main problem that we are trying to solve with a generative model is how to generate new data points from the unknown distribution, given that we know the data set $\mathbb{X}$.

In this text we will discuss generative models that are needed to understand VQ-VAE. There are two types of generative models that are related to VQ-VAE: the first is autoregressive models and the second is latent based models such as variational autoencoder (VAE). Both of these models are important to understand VQ-VAE.

3.1 Autoregressive Models

This section is partly follows the ideas from [10]. Assume our data set $\mathbb{X}$ consists of n-dimensional vectors $\mathbf{x} = (x_0, x_2, \ldots, x_{n-1})$ where each dimensions can take k categorical values from the set $\mathbb{C} = \{0, 1, \ldots, K-1\}$ such that $x_i \in \mathbb{C}, \quad i = 0, 1, \ldots, n$. Let $\mathbf{x}_{<i}$ represents the set $\{x_0, x_1, \ldots, x_{i-1}\}$. We can then use the chain rule to write the distribution of $\mathbf{x}$ as follows

$$p(\mathbf{x}) = \prod_{i=1}^{n} p\left(x_i \mid x_1, x_2, \ldots, x_{i-1}\right) = \prod_{i=1}^{n} p\left(x_i \mid \mathbf{x}_{<i}\right) \tag{3.1}$$

and, thus, the likelihood function would be

$$\log p(\mathbf{x}) = \sum_{d=1}^{n} \log p\left(x_i \mid \mathbf{x}_{<i}\right) \tag{3.2}$$

If we are able to parameterize a seperate function for each conditional such that it only takes as inputs $\mathbf{x}_{<i}$ and output a true discrete distribution for x_i, then we can write a true likelihood function. Outputing a discrete distribution over the categories is described in 2.2. In [10], they called this property the autoregressive property. Models which satisfies this property are called autoregressive models.

However, having a seperate parametrised model for each conditional distribution is impractical because we need a model for each dimension, considering the complexity of modern deep learning models.

One solution to this problem is parameter sharing [18]. In fact, the two autoregressive

models that we will mention in this text use only one neural network to parameterise all the conditionals. For each conditional distribution, $p(x_i|x_{<i})$, the model (e.g. a neural-network) perform a mapping from the vector $\mathbf{x}_{<i}$ to a discrete distribution over the categories in $\mathbb{C}$ as described in 2.2. Again, using equation 3.2 we can obtain a likelihood function and train it using maximum-likelihood estimation.

However, this brings us to the first issue with AR models. As you can see from equation 3.1, the choice of ordering the dimensions or features is left up to the practitioner. Obviously, taking a wrong order of dimensions can make a difference. For example, in real-life images, far apart pixels are almost independent. In [10], they suggested training over all possible orderings by sampling an ordering before each update.

3.2 MADE: Masked Autoencoder for Distribution Estimation

Masked Autoencoder for Distribution Estimation For simplicity (MADE) was introduced in [10], and we will follow a similar description theirs. Their description help explain why treating the output of the model as probabilities, such as in the cross entropy loss functions, are not true probabilities.

Let the number of possible categories equal two, $k = 2$, $x \in 0, 1^n$. In [10], they have shown that a single autoencoder with MLP architecture can be used to parameterise all the conditional distributions in 3.2.

In our simple case of $k = 2$, we can design an autoencoder that outputs a vector $\mathbf{p} = (p_0, p_1, \ldots, p_n)$ of the same dimensions as the input data such that each p_i is a probability that corresponds to $p(x_i = 1 \mid \mathbf{x}_{<i})$. This can be done by using a sigmoid activation function at the last layer, see 2.1.1. If we are to implement the cross-entropy loss function which is what usually done with categorical data we get

$$\ell(\mathbf{x}) = -\sum_{i=1}^{n} x_i \log p_i + (1 - x_i) \log (1 - p_i)$$

Notice that if p_i is treated as a probability, then this loss is the negative log-likelihood function. This means the probability of $\mathbf{x}$ based on this model is

$$P(\mathbf{x}) = \prod_{i=1}^{n} x_i p_i + (1 - x_i) (1 - p_i)$$

However, this is not a true distribution because $\sum_{x \in \mathbf{X}} p(x)$ is not one.

To make it a true distribution we need to satisfy the chain rule. To achieve that, each output dimension p_i should only depends on $\mathbf{x}_{<i}$ thus p_i would correspond to $p(x_i = 1 \mid \mathbf{x}_{<i})$ and $(p_i - 1)$ would correspond to $p(x_i = 0 \mid \mathbf{x}_{<i}$.

To only allow information flowing from $\mathbf{x}_{<i}$, we zero out any computation resulting from the the input dimensions $x_i, x_{i+1}, \ldots, x_n$. This can be done by multiplying the weights with a binary mask matrix M. The element of the mask matrix is zero at the positions corresponding to the unwanted computations. Rewriting equation 2.1 of the MLP from section 2.1.1 as

$$h^l(x^{l-1}) = a(b^l + M^l \odot W^l x^l) \tag{3.3}$$

Different choice of the mask M results in different models. The original paper [10] delve into the details of designing these masks. However, the idea of using masks to cut off computations that violates the chain rule is what we need to study further related models.

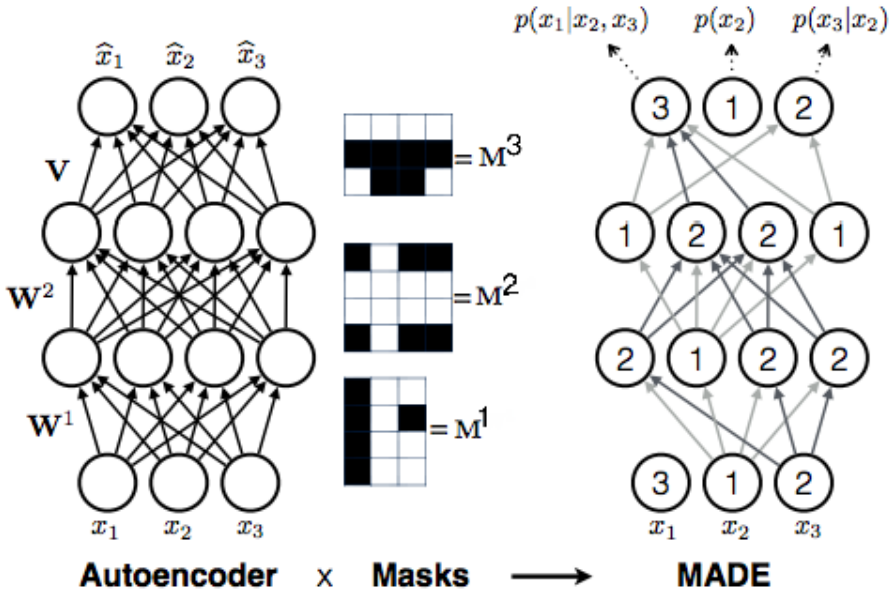

Figure 3.1: On the left is an autoencoder with three hidden layers. In the middle, three masks in which a black colour is zero and white is one. On the right, we see how applying these masks to each aligned layer cut the computational paths. For example, M^{W^1} is applied to the first hidden layer to cut x_1 completely and cuts x_3 for the second neuron in the second layer. Overall, the uncut computational paths corresponds the probability chain rule such that the probability of $\mathbf{x}$ can be written as $P(\mathbf{x}) = P(x_2)P(x_3 \mid x_2)P(x_1 \mid x_2, x_3)$ [10]

.

Finally, this description can be generalized to data with more categories by mapping

the data $\mathbf{x}$ into their one-hot encoding. A one-hot encoding of the feature x_i is a k-dimensional vector $u = (u_0, u_1, \ldots, u_{k-1})$ that is one at u_i and zero everywhere else. The probability of x_i in this model would be the dot product with the vector representing the discrete distribution, as described in section 2.2.

3.3 PixelCNN

We have seen how MADE applies masks to satisfy the autoregressive property. PixelCNN solves a similar problem but for CNNs. To explain the basic concept, we will assume that each data point $\mathbf{x}$ belongs to $R^n xn$ which is a 2d matrix of n columns and n rows. Each element $x_{i,j}$ of $\mathbf{x}$ is k categorical.

PixelCNN [23] tries to parametrise each of the following conditionals $p(x_{i,j}|x_{\{<i,<j\}})$ where $x_{<i,<j} = \{x_{u,t} \mid u < i, t < j\}$. Thus it needs to output $n \times n$ categorical distributions, see section 2.2.

Note that we did the same thing with MADE example in the previous section except that the categories are two and thus one probability $p(x_i = 1)$ output is enough, since $p(x_i = 1) = 1 - p(x_i = 1)$.

The computational path of each output distribution of $\mathbf{x}_{i,j}$ must depend only on $\mathbf{x}_{\{<i,<j\}}$.

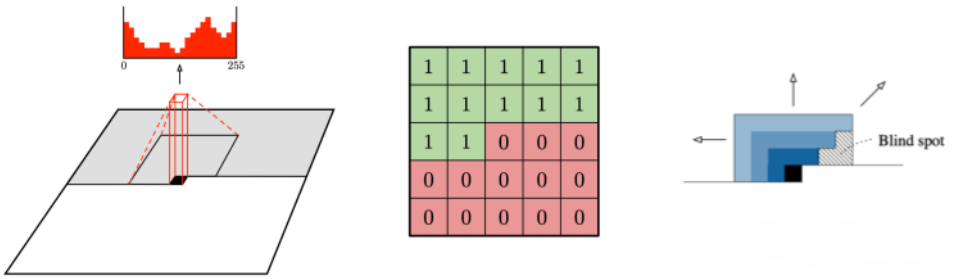

Figure 3.2: **Left:** A visualization of generating the discrete distribution using the receptive field caused by the masked kernel. **Middle:** Type A mask of size 5. **Right:** The receptive field of the described PixelCNN caused by the introduced masks have blind spots in nearby pixels [28]

.

To achieve that we zero out any computation done with any input that is not from the set $\mathbf{x}_{\{<i,<j\}}$. This is done again using binary masks M_c^l, where l is the layer number and c is the channel number, see section 2.1.2. The mask M_c^l is multiplied with the corresponding kernel element-wise.

The mask of the first layer M_c^0 is the same size as the corresponding kernel which is

$m \times m$ size. It can be defined as follows

$$M_c^0[i,j] = \begin{cases} 1 & \text{if } i < \lceil \frac{n}{2} \rceil \text{ or } i = \lceil \frac{n}{2} \rceil,\ j < \lceil \frac{n}{2} \rceil \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

In the original paper this is called mask type A. However, this mask can not be used again in other layers, instead mask type B is applied. Mask type B is similar to type A except it is set to one at the center $M_c^0[i,j]$ instead of zero,

$$M_c^l[i,j] = \begin{cases} 1 & \text{if } i < \lceil \frac{n}{2} \rceil \text{ or } i = \lceil \frac{n}{2} \rceil,\ j \leq \lceil \frac{n}{2} \rceil \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

This is enough to satisfy the autoregressive property. However, type B can be defined differently if we want to achieve conditioning between colors in images, see [23, 28] for more details. Moreover, as you can see from figure 3.3, some nearby pixels are missing from the receptive field, to obtain a better receptive field see [23] for details.

3.4 VAE

Another way to model the data distribution is to assume that their are unobserved variables that influence the data. These variables are called latent variables.

Let $\mathbf{z}$ be a latent variable that is distributed by $P(\mathbf{z})$ is called the prior. Instead of defining an explicit distribution of the data directly, we define a conditional $p_\phi(\mathbf{x} \mid \mathbf{z})$ which is the distribution of data conditioned on the latent variable and it is called the decoder. ϕ is the learnable parameters of whatever model we chose. If we defined $p(\mathbf{z})$ to be some fixed distribution, then this is enough to define the likelihood function as

$$\ell(x) = \log \mathbb{E}_{p(\mathbf{z})}\, p_\phi(\mathbf{x}|\mathbf{z}) \tag{3.6}$$

where the expectation is over the distribution of the latent variable.

Example 3.1 A Gaussian mixture model (GMM) consists of a discrete latent variable z which takes values from $\{0, 1, \ldots, K-1\}$. The latent is distributed via a uniform prior

$P(z)$. Finally, it consists of k different conditional distributions defined as follows

$$P_{\mu_z,\Sigma_z}(\mathbf{x} \mid z) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_z)^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\mu_z)\right)}{\sqrt{(2\pi)^k|\Sigma_z|}} \tag{3.7}$$

Where μ_z is the mean of the Gaussian and Σ_z is the covariance matrix.

Following equation 3.6, the log-likelihood function becomes

$$\ell(\phi \mid \mathbf{x}) = \log \sum_z P(z) P_{\mu_z,\Sigma_z}(\mathbf{x} \mid z),$$

Where ϕ represents the set of all mean and covariance matrices defined by the conditionals. Our goal is to learn the parameters ϕ that maximize the log-likelihood.

Note that the distribution $P(z|x)$ can be viewed as the responsibility of the Gaussian $\mathbf{z}$ to generate instance x.

Notice that if the latent variable $\mathbf{z}$ is multidimensional and can take large number of values, then both evaluating and differentiating the expectation in 3.6 can become intractable.

We will describe the solution for this intractability from the perspective of importance sampling, see more in [4]. Again, let $\mathbb{X}$ be the data set, and $\mathbf{x}$ be a data instance which is sampled from the unknown distribution. Note that it is wronge to approximate the expectation by sampling from $p(\mathbf{z})$, because it is hard to sample $\mathbf{z}$ that carries information[1] about the already sampled $\mathbf{x}$, if $\mathbf{z}$ can take large number of values.

It is not made clear yet, but it seems that we need to sample from $P(\mathbf{x} \mid \mathbf{z})$ which can be written as

$$P(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z})}{P(\mathbf{x})}$$

However, $P(\mathbf{x})$ is intractable, because, again, the log-likelihood 3.6 is intractable. We will call $P(\mathbf{x} \mid \mathbf{z})$ the true posterior.

Instead, we introduce a different parameterised distribution $q_\theta(\mathbf{z}|\mathbf{x})$ that will be used to sample latent variable values. This distribution is called the approximate posterior.

[1] From an information theory perspective, we need to reduce the the uncertainty about $\mathbf{x}$ given $\mathbf{z}$

Following [4], the method of replacing $P(\mathbf{z})$ with $q_\theta(\mathbf{z}|\mathbf{x})$ in equation 3.6 is as follows

$$\begin{aligned}
\log P(\mathbf{x}) &= \log \mathbb{E}_{p(\mathbf{z})} \frac{q_\theta(\mathbf{z}|\mathbf{x})}{q_\theta(\mathbf{z}|\mathbf{x})} p_\phi(\mathbf{x}|\mathbf{z}) \\
&= \log \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})} \frac{p(z)}{q_\theta(z|x)} p_\phi(x|z) \\
&\geq \mathbb{E}_{q_\theta(z|x)} \log \frac{p(z)}{q_\theta(z|x)} p_\phi(x|z) \\
&= \underbrace{\mathbb{E}_{q_\theta(z|x)} \left[\log p(z) - \log q_\theta(z|x) + \log p_\phi(x|z)\right]}_{ELBO}
\end{aligned} \tag{3.8}$$

Where the inequality is the Jensen inequality. The last term is called the evidence lower bound ($ELBO$).

From the deriviation we can see that maximizing the $ELBO$ corresponds to maximizing the log-likelihood.

The gap between the $ELBO$ and the log-likelihood is due to Jensen inequality. Note that Jensen inequality is an equality if and only if the function inside the expectation is constant or affine [6]. However, the log is convex, and thus we achieve an equality if and only if the term is constant [14].

As it was pointed out in [14], to obtain a constant, $q_\theta(\mathbf{z} \mid \mathbf{x})$ should be directly proportional to $p(\mathbf{z})p_\theta(\mathbf{x} \mid \mathbf{z})$ (meaning their division equals a constant). Dividing $p(\mathbf{z})p_\theta(\mathbf{x} \mid \mathbf{z})$ by the unknown constant $P(\mathbf{x})$ (constant in terms of the expectation) we obtain the true posterior $P(\mathbf{x} \mid \mathbf{z})$, therefore, the approximate posterior should be directly proportional to the true posterior which in turns means they should be equivalent because they are distributions of the same random variable $\mathbf{z}$.

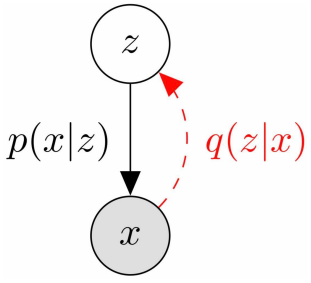

Figure 3.3: Bayesian network representation of the VAE [30]

.

To measure the difference between the true posterior and the approximate posterior $q_\theta(z|x)$ we use the Kullback–Leibler divergence ($\mathrm{D_{KL}}$) which is a well-known approach

to derive the $ELBO$ and it is as follows

$$\begin{aligned}
\mathrm{D}_{\mathrm{KL}}\left[q_\theta(z|x)\|P(z|x)\right] &= \mathbb{E}_{q_\theta(z|x)}\left[\log q_\theta(z|x) - \log p_\phi(z|x)\right] \\
&= \mathbb{E}_{q_\theta(z|x)}\left[\log q_\theta(z|x) - \log \frac{p_\phi(x|z)p(z)}{P(x)}\right] \\
&= \mathbb{E}_{q_\theta(z|x)}\left[\log q_\theta(z|x) - \log p_\phi(x|z) + \log p(z) - \log p(x)\right] \\
&= \mathbb{E}_{q_\theta(z|x)}\left[\log q_\theta(z|x) - \log p_\phi(x|z) + \log p(z)\right] - \log P(x)
\end{aligned} \tag{3.9}$$

where the last equality is because $\log p(x)$ is a constant in terms of the expectation, which gives

$$\log p(x) = \mathbb{E}_{q_\theta(z|x)}\left[\log q_\theta(z|x) - \log p_\phi(x|z) + \log p(z)\right] - \mathrm{D}_{\mathrm{KL}}\left[q_\theta(z|x)\|p(z|x)\right] \tag{3.10}$$

This means that the error gap between the $ELBO$ and the true $\log P(x)$ is the D_{KL} divergence between the approximate posterior and the true posterior.

The $ELBO$ can be simplified as follows

$$\begin{aligned}
ELBO &= \mathbb{E}_{q_\theta(z|x)}\left[\log p(z) - \log q_\theta(z|x) + \log p_\phi(x|z)\right] \\
&= \underbrace{\mathbb{E}_{q_\theta(z|x)}\left[\log p_\phi(x|z)\right]}_{\text{reconstruction error}} + \underbrace{\mathrm{D}_{\mathrm{KL}}\left[q_\theta(z|x)\|\mathbf{p}(z)\right]}_{\ell_{\mathrm{D}_{\mathrm{KL}}}}
\end{aligned} \tag{3.11}$$

Where the first term is the reconstruction error and the last is divergence between the approximate posterior and the prior.

3.4.1 Repramatrisation Trick

We introduced the approximate posterior to perform sampling for our optimization problem. However, if we to use sampling to replace the expectation of the reconstruction loss in equation 3.11 by the empirical expectation, we lose $q_\theta(z|x)$ and thus the derivative with respect to θ.

One solution is to use what is called the reparameterization trick. The trick is done by rewriting the random variable $\mathbf{z}$ as a function of another random variable with a fixed distribution that is easy to sample from. Let this variable be y which is distributed by the fixed distribution $p(y)$ such that

$$z = g_\theta(y, x)$$

This means that we can sample from $q_\theta(z|x)$ by sampling y from $p(y)$ and then $z = g_\theta(z, x)$ is a sample from the approximate posterior.

The reconstruction loss can be rewritten as

$$\ell_{reconstruction-loss} = \mathbb{E}_{p(y)} \log q_\phi(g_\theta(e, x)|x)$$

With that, we can approximate the expectation by sampling from $p(y)$ and take the gradient over θ.

Unfortunately, this method is not applicable for discrete distributions [15]. One solution for that is to approximate the discrete distribution with a continuous one as it was done in [3, 15]. However, we will not discuss this in this text. Instead, we will use vector quantization to solve this problem as we will see in section 5.

4 Clustering

Clustering algorithms partition the data into sets called clusters with the goal that instances within a single cluster are more similar to each other than instances belonging to different clusters [7]. Similarity between data instances can be based on distance measures such as the Euclidean distance, the shorter the distance the more similar they are[2]. Clustering can be fuzzy in such away that a single element can belong to multiple clusters with different degree of certainty.

In machine learning, clustering algorithms are unsupervised learning algorithms mainly because they do not require data labels. In this chapter we are interested in clustering algorithms that represent each cluster with a vector called prototype or cluster center. These algorithms are also called vector quantization (VQ) which can be viewed as a compression algorithms in which the data is encoded into a finite set of vectors.

Let k be the number of clusters that we wish to find. Assume our data belong to a d-dimensional vector space $\mathbb{R}^d$ and let $\mathbb{C} = \{0, 1, \ldots, k-1\}$ be a set of indices of cardinality k in which each number represents a cluster. Hard clustering maps the input data to the set $\mathbb{C}$ by assigning each instance to a single cluster, forming a many-to-one mapping $\gamma : R^d \to \mathbb{C}$ which is a discrete mapping. On the other hand, soft-clustering introduces scores that evaluate how likely for a data instance $\mathbf{x}$ to belong to a certain cluster $z \in \mathbb{C}$. These scores can be called membership scores. Of course, soft clustering can perform hard clustering by choosing the cluster with the maximum score.

4.1 Vector Quantization

Again let k be the number of clusters we wish to find. VQ algorithms perform clustering by introducing a finite set of vectors $\mathbf{w} = \{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{k-1}\}$. The set of vectors $\mathbf{w}$ is called the codebook and each vector is called a prototype. Each prototype w_z belongs to the space of the input data and z belongs to the set $\mathbb{C} = \{0, 1, \ldots, k-1\}$. Each prototype w_z corresponds to a single cluster z.

Notice that any choice of prototypes will partition the data space into subregions called the Voronoi polyhedra. In [8], they are defined as follows

$$R_i = \{x \mid \mathrm{d}(x, w_i) \leq \mathrm{d}(x, w_j) \quad \forall i \neq j\}, \quad i, j \in \mathbb{C}.$$

[2] This is an informal way to describe similarity, see [22]

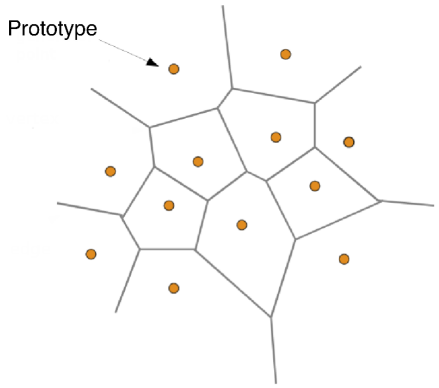

Figure 4.1: The circles corresponds to 2d prototypes. The black boundaries corresponds to an equal distance between at least two prototypes. [8]

.

A distortion measure is introduced to quantify assigning an instance $\mathbf{x}$ to a cluster z represented by the prototype $\mathbf{w}_z$. In this text, the distortion measure is a distance measure $\mathrm{d}(\mathbf{x}, \mathbf{w}_z)$ defined on the data space. Through out the text, the distance will be the Euclidean distance, $\mathrm{d}(\mathbf{x}, \mathbf{w}_z) = \|\mathbf{x} - \mathbf{w}_z\|^2$.

Let the mapping $\gamma(x) = \mathrm{argmax}_{z \in \mathbb{C}}\, dis(x, w_z)$ be called the quantizer encoder which encodes the data x into a symbol from $\mathbb{C}$ [12]. That is why vector quantization is a form of compression in which we map the data into code symbols from $\mathbb{C}$. Lastly, Let the mapping $\beta(x) = \mathbf{w}_{\gamma(x)}$ be called the quantizer which maps a data instance $\mathbf{x}$ to the nearest prototype.

4.1.1 A Mutual Information Loss

In VQ the general goal is to choose a cluster assignments that minimize the expected distance $\mathrm{d}(\mathbf{x}, \mathbf{w}_z)$ [12]. We will let z to be a discrete random variable that only takes values from the cluster indices $\mathbb{C}$. For any instance $\mathbf{x}$ the a clustering algorithm outputs a probability $p_{\mathbf{w}}(z|\mathbf{x})$ for each cluster $z \in \mathbb{C}$. We call it the responsibility in analogy with GMM [19, 30]. The higher the probability the more likely that $\mathbf{x}$ belongs to cluster z.

In this model the expected $\mathrm{d}(x, w_z)$ is

$$\mathrm{E} = \mathbb{E}_{\mathbf{x},\mathbf{z}}\, \mathrm{d}(\mathbf{x}, \mathbf{w}_z) \approx \sum_{x \in X} \sum_{z \in \mathbb{C}} P_{\mathbf{w}}(z|x)\, \mathrm{d}(\mathbf{x}, \mathbf{w}_z) \tag{4.1}$$

While clustering algorithms do not need to be probabilistic, we will see that some VQ algorithms can be reduced to the loss in equation 4.1.

Note this is a generalization of hard-clustering, because we can think of any hard clustering algorithm as if it outputs a deterministic probabilities $p(z|x) = \mathbf{1}_{\gamma(x)=z}$ which means a probability of one is given to the cluster $\gamma(x)$. Moreover, this type of clustering algorithms can be reduced to a hard clustering algorithm by defining the mapping $\gamma(\mathbf{x}) = \text{argmax}_{z\in\mathbb{C}}\, p(z|\mathbf{x})$.

Note that the Euclidean distance, can be viewed as the log of a conditional Gaussian with identity covarinace, ignoring constants. This Gaussian is conditioned on z. So we can view equation 4.1 as the cross entropy $\mathscr{H}(\mathbf{x} \mid z)$ and in consequence the mutual information between z and $\mathbf{x}$ since $\mathbf{x}$ has an unknown fixed distribution. Therefore optimizing equation 4.1 corresponds to finding the posterior that maximizes the mutual entropy between $\mathbf{x}$ and z. That is why we will call equation 4.1, the VQ mutual information loss. See [20]

4.1.2 K-means

K-means is a well-known hard clustering algorithm. Again let $\gamma(x) = argmin_{z\in C}\, \mathrm{d}(x, w_z)$. We can define the following posterior

$$P_{\mathbf{w}}(z|x) = \mathbf{1}_{z=\gamma(x)}$$

Which is a deterministic posterior that is one at $\mathbf{z} = \gamma(x)$. The k-means objective function can be written by substituting this posterior in equation 4.1

$$\mathrm{E} = \sum_{\mathbf{x}\in X}\sum_{z\in\mathbb{C}} \mathbf{1}_{z=\gamma(\mathbf{x})}\, \mathrm{d}(\mathbf{x}, \mathbf{w}_z) = \sum_{\mathbf{x}\in\mathbb{X}} d(\mathbf{x}, \mathbf{w}_{\gamma(\mathbf{x})}) \tag{4.2}$$

Taking the gradient over E gives the Lloyd's and MacQueen k-means algorithm [21]. However, the problem with this algorithm is that $\ell(x)$ has many local minima [21].

4.2 Neural Gas

Neural gas [21] defines a ranking function $r(\mathbf{x}, z)$ called "the neighbourhood ranking" which outputs the rank of the prototype $\mathbf{w}_z$ relative to all other prototypes in terms of the distortion measure $\mathrm{d}(x, \mathbf{w}_z)$. $r(\mathbf{x}, z)$ permutes the set $\mathbb{C}$ such that $r(\mathbf{x}, i) = 0$ if $\mathbf{w}_i$ is the closest prototype, $r(\mathbf{x}, i) = 1$ if w_i is the second closest and so on until $\mathbf{w}_i$ is the furthest then $r(\mathbf{x}, i) = k - 1$.

For a data set X and a codebook $\mathbb{C}$, [7] defines the objective function of neural gas as

follows

$$\mathrm{E} = \sum_{\mathbf{x}\in\mathbb{X}} \sum_{z\in\mathbb{C}} h_\lambda(r_{\mathbf{w}}(\mathbf{x}, z))\, \mathrm{d}(\mathbf{x}, \mathbf{w}_z) \tag{4.3}$$

such that $h_\lambda(u) = e^{\left(\frac{-u}{\lambda}\right)}$.

Let $\mathbf{D} = \sum_{z\in\mathbb{Z}} h_(r_{\mathbf{w}}(x, z))$ be a normalization factor. Notice that $\mathbf{D}$ is a constant. We can thus define the following discrete posterior

$$p_{\mathbf{w}}(z|x) = \frac{h_\lambda(r_w(x, z))}{\mathbf{D}}$$

Thus the neural gas algorithm can be reduced to optimizing the objective function in equation 4.1. In [21], they have shown that this constitute a differentiable objective function.

4.3 Fuzzy c-Means

Fuzzy c-means is a soft-clustering algorithm. For each instance $\mathbf{x}$, it outputs membership scores for each prototype that measure how likely an instance $\mathbf{x}$ to belong to each one. The membership score of an instance $\mathbf{x}$ to belong to the cluster z is $\mathbf{w}_z(\mathbf{x})$ and is defined as follows

$$\mathbf{w}_z(\mathbf{x}) = \left(\sum_{c\in C} \left(\frac{\mathrm{d}(\mathbf{x}, \mathbf{w}_z)}{\mathrm{d}(\mathbf{x}, \mathbf{w}_c)} \right)^{\frac{2}{m-1}} \right)^{-1} \tag{4.4}$$

Where $m \geq 1$. m is a hyperparameter and it is called a fuzzifier. The fuzzifier control the "softness" of the membership in such away that the limit of the memberships approaches a deterministic distribution as m approaches one and thus the algorithm becomes similar to k-means if m is very close to one.

The objective function of fuzzy c-means is

$$\mathrm{E}(x) = \sum_{x\in X} \sum_{z\in\mathbb{C}} w_z(x)^m d(x, w_z)\ , \tag{4.5}$$

which is differentiable [27].

The objective functions can not be reduced to equation 4.1, because $\sum_z w_z(x)^m \neq 1$ and a normalization factor would not be a constant.

However, we will define two alternative posteriors for experimentation purposes. First, we define the posterior

$$p_{\mathbf{w}}(z|x) = \mathbf{w}_z(x) \tag{4.6}$$

Second, if we divided $w_z(x)^m$ by a a normalization factor $\mathbf{D}_{\mathbf{w}}(x) = \sum_{z \in \mathbb{C}} w_z(x)^m$, then we can define a discrete posterior as follows

$$p_{\mathbf{w}}(z|x) = \frac{1}{\mathbf{D}_{\mathbf{w}}(x)} \left(\sum_{c \in C} \left(\frac{\mathrm{d}(x, w_z)}{\mathrm{d}(x, w_c)} \right)^{\frac{2}{m-1}} \right)^{-m} \tag{4.7}$$

If we substitute these posteriors in equation 4.1, we obtain objective functions that are different from the original fuzzy c-means. Again, these are just for experimenting on different posteriors.

4.4 VQ with Large Number of Prototypes

Vector quantization as a compression technique where we encode a data point $\mathbf{x}$ to one element from $\mathbb{C}$ or the codebook W. We can lose a lot of information by having a small number of prototypes (or small number of clusters k), especially if we have complex data such as real life images.

In every VQ objective function we encountered in this text, there is a summation over the set $\mathbb{C}$, which corresponds to the expectation over the posterior in equation 4.1. If $\mathbb{C}$ has a large cardinality the objective function would be intractable.

One solution to this problem is to partition the dimensions of the input data into subsets. We can then assume these subsets are independent and thus apply vector quantization independently on each subspace. We also can share the same prototypes between all these subspaces, thus we obtain a tractable algorithm in both time and space (space corresponds to memory allocation in practice).

For example, Let the input data belong to R^d and $d = c \times d'$. Choose k prototypes $\{\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}\}$ such that each prototype $\mathbf{w}_i$ belongs to R^c, then we can have k^d number of prototypes in the R^d space

$$\mathbf{w} = \{\left(\mathbf{w}_{z,0}, \ldots, \mathbf{w}_{z,k-1}\right) \mid \mathbf{w}_{z,i} \in \mathbf{w} \quad \forall i \in C\}$$

which also corresponds to the discrete representation of categorical vectors that belong to $\mathbb{C}^k = \{(z_0, z_1, \ldots, z_{k-1}) \mid z_i \in C, \quad \forall i \in C\}$.

Note that this will compress a data of d-dimensional data into categorical data of d' dimensions where each dimension can take k values only. This is important because many models such as PixelCNN or MADE require categorical data

For example, images of 16×16 dimensions can be turned into 4×4 dimensions by applying VQ to orthognal subspaces of four dimensions each (e.g. applying VQ to four pixels independently).

One flaw with this approach is the assumption that the subspaces are independent. To obtain a better compression we need to drop this assumption which is what VQ-VAE does.

5 VQ-VAE

Using discrete functions in Machine Learning can be problematic. They are basically not differentiable and this makes implementing gradient optimization methods a hurdle. We have seen that clustering is a mapping from the data space to a finite set of clusters and thus this mapping is discrete and not differentiable, yet we were able to implement gradient descent optimization using some vector quantization models. In this chapter we will discuss VQ-VAE which is a generative model that take use of vector quantization methods.

VQ-VAE is a latent variable model that consists of an encoder $\mathrm{z}_\theta(x)$, a VQ inspired posterior $q_{\mathbf{w}}(z \mid \mathrm{z}_\theta(\mathbf{x}))$ that is parametrised by a codebook $\mathbf{w}$, and a decoder $p_\phi(x \mid z)$. Note if the chosen VQ algorithm does not permit a posterior, we opt for a determnestic posterior on the output of the quantizer $\beta(x)$. Both ϕ and θ are learnable parameters. The encoder $\mathrm{z}_\theta(x)$ maps from the input data to what we will call a continuous latent space.

Let z be a latent variable that takes values from $\mathbb{C} = \{0, 1, \ldots, k-1\}$ where k is the number of prototypes. The posterior $q_{\mathbf{w}}(z \mid \mathrm{z}_\theta(x))$ is parameterized by a codebook $\mathbf{w} = \{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{k-1}\}$ that consists of vectors called prototypes. Each prototype belongs to the continuous latent space. Similar to VQ we define a distortion measure over the continuous latent space $\mathrm{d}(\mathrm{z}_\theta(x), \mathbf{w}_z)$ which will be the Euclidean distance between $\mathrm{z}_\theta(x)$ and the prototype $\mathbf{w}_z$.

Also similar to VQ, we define a quantizer encoder $\gamma(x)$ maps from the continuous latent space to the set $\mathbb{C}$, $\gamma(x) = \mathrm{argmin}_{z \in \mathbb{C}} \mathrm{d}(\mathrm{z}_\theta(x), \mathbf{w}_z)$.

Note that this is a simplification that makes discussion easier, in practice, we will have multiple latent variables, or z is a random categorical vector that belongs to $\mathbb{C}^m$ such that m is large enough to achieve a reasonable compression in the latent space as described in 4.4.

As it has been described in [24], VQ-VAE consists of two main stages: The first stage is compressing the input data into a smaller discrete representation in the discrete latent space. The second stage is learning the distribution of this discrete representation which is called learning the prior $p(z)$. Once we have learned the prior $p(z)$, we can sample from $p(z)$ and pass to the encoder to sample $\mathbf{x}$, thus it is a generating model. We will go through all of this in more details.

5.1 Stage 1: Compression

Similar to an autoencoder, the main goal is to optimize the reconstruction loss $\ell(x) = \log p_\phi(x \mid \beta(x))$. However the quantizer is a discrete mapping and we can not backprobogate to the encoder $\mathrm{z}_\theta(\mathbf{x})$. In [29], they proposed to use the straight-through gradient estimator which is simply to copy the gradient with respect to the quantizer directly to the encoder.

VQ-VAE take use of the stop-gradient operator, represented by sg. This operator stops the gradient from being evaluated on its argument [29]. In the backpropagation algorithm, it is treated as the identity during forward pass, while it is treated as a constant during the backward pass, thus no need to flow through it.

We can rewrite the straight-through estimator for the reconstruction error in terms of sg as follows

$$\ell(x) = \log p_\phi(x \mid \mathrm{z}_\theta(x) + \mathrm{sg}[\mathbf{w}_{\gamma(x)} - \mathrm{z}_\theta(x)])$$

With that we pass the $w_{\gamma(x)}$ to the decoder but we flow through $\mathrm{z}_\theta(x)$ in the backward pass. However, the prototypes are fixed (and so is $\beta(x)$) because they do not receive gradient information [29].

Moreover, the quantizer introduces an error that can be measured by $\mathrm{d}(\mathrm{z}_\theta(x), \mathbf{w}_{\gamma(x)}) = \|\mathrm{z}_\theta(\mathbf{x}) - \beta(x)\|^2$. Therefore, besides optimizing the reconstruction error, we need to minimise the expectation of $\mathrm{d}(\mathrm{z}_\theta(\mathbf{x}), \mathbf{w}_{\gamma(x)})$. Given the VQ inspired posterior $q_\mathbf{w}(z \mid \mathrm{z}_\theta(\mathbf{x}))$, the expectation would be

$$\mathbb{E}_{q_\mathbf{w}(z|\mathrm{z}_\theta(\mathbf{x}))} \|\mathrm{z}_\theta(x) - \mathbf{w}_z\|^2 \tag{5.1}$$

However, this expectation sends gradient information to the encoder $\mathrm{z}_\theta(\mathbf{x})$ which already receives gradient information from the reconstruction loss. We simply need to change the update "speed" of the encoder parameters caused by this expectation. One solution is changing the step size used by the gradient descent algorithm for this expectation on the encoder parameters. Another solution is to use the sg operator to split the expectation into two losses. One loss is called the commitment loss, which is responsible for updating the encoder parameter to commit to the prototypes. It is derived by simply applying sg to the prototypes

$$\alpha \sum_z q_{\mathrm{sg}(\mathbf{w})}(z \mid \mathrm{z}_\theta(x)) \|\mathrm{z}_\theta(x) - \mathrm{sg}(\mathbf{w}_z)\|^2 \tag{5.2}$$

Where α is to control the update "speed" of the encoder parameters. The commitment loss help the output of $z_\theta(X)$ from growing arbitrarily large, which corresponds to large polyhedra areas and thus bad pass-through estimator as $\|z_\theta(x) - \mathbf{w}_{\gamma(x)}\|$ becomes large [29].

The other loss is derived by applying sg to the encoder as follows

$$\eta \, \mathbb{E}_z \, q_{\mathbf{w}}(z \mid x) \|\mathrm{sg}(z_\theta(x)) - \mathbf{w}_z\|^2 \tag{5.3}$$

where η is a hyperparameter. This loss will be called the VQ loss because it treats the output of $z_\theta(x)$ as the input data in a VQ loss (by not updating them).

The overall loss function becomes

$$\begin{aligned} \ell(x) &= \log p_\phi(x \mid \beta(x)) \\ &\quad + \alpha \, \mathbb{E}_{q_{\mathrm{sg}(\mathbf{w})}(z|z_\theta(x))} \|z_\theta(x) - \mathrm{sg}(\mathbf{w}_z)\|^2 + \eta \, \mathbb{E}_{q_{\mathbf{w}}(z|x)} \|\mathrm{sg}(z_\theta(x)) - \mathbf{w}_z\|^2 \end{aligned} \tag{5.4}$$

The model is very sensitive to α because it will interfere with the gradient information from the reconstruction error. Notice that if α and η are equal then the commitment loss and the VQ loss add up to be

$$\alpha \sum_z q_{\mathbf{w}}(z \mid x) \|z_\theta(x) - \mathbf{w}_{\gamma(x)}\|^2.$$

Finally, we can create different variants of this model via simply introducing different discrete posteriors. We can replace the posterior $q_{\mathbf{w}}(z \mid x)$ with the neural gas posterior to obtain our implementation of neural gas with VQ-VAE. Posteriors defined in section 4.3 in both equation 4.5 and 4.6 can be applied to this loss function.

Notice also that the reconstruction error and the expectation of $\|z_\theta(x) - \mathrm{sg}(w_z)\|^2$ are done on different spaces with possibly different losses, thus the choice of the decoder $p_\phi(x \mid z_\theta(x))$ will affect the right choice of α. Informally, it is hard to balance the gradient information coming from the decoder with the gradient obtained from the expectation at the continuous latent space.

5.1.1 Vanilla VQ-VAE

VQ-VAE was first introduced in [29], where they defined the posterior as follows

$$q_{\mathbf{w}}(z \mid \mathbf{x}) = \begin{cases} 1 & z = \operatorname{argmin}_{j \in \mathbb{C}} \|e(x) - w_j\| \\ 0 & \text{otherwise} \end{cases}$$

which corresponds to the k-means posterior described in 4.1.2. Substituting the loss of the k-means in equation 5.8, we arrive at the original VQ-VAE loss function

$$\ell(x) = \log p\left(x \mid \beta(x)\right) + \alpha \|\mathrm{z}_\theta(x) - \mathrm{sg}(\mathbf{w}_z)\|^2 + \eta \|\mathrm{sg}(\mathrm{z}_\theta(x)) - \mathbf{w}_z\|^2 \tag{5.5}$$

We will call this the vanilla VQ-VAE. Note that in [29] suggests changing the last term (the VQ loss) to introduce different VQ models while leaving the commitment loss (the second term) without change. In our experiments we found our approach to perform better with neural gas.

5.1.2 VQ-VAE with Fuzzy c-Means

The problem with fuzzy c-means is that it can not be reduced to optimizing a true expectation over a discrete posterior. The loss function of the fuzzy c-means is

$$\mathrm{E}(x) = \sum_{x \in X} \sum_{z \in \mathbb{C}} w_z(x)^m d(x, w_z) \tag{5.6}$$

Notice the factor $w_z(x)^m$ that multiplies $d(x, w_z)$. Because $\sum_{z \in \mathbb{C}} w_z(x)^m$ is not constrained to a constant, this summation value will be changing during training. Thus the scaling that we control with the hyperparameter α is chaotic and controlling it with one hyperparameter seems hard.

On the other hand, using this loss

$$\mathrm{E}(x) = \sum_{x \in X} \sum_{z \in \mathbb{C}} w_z(x) d(x, w_z) \tag{5.7}$$

provides good results, because $\sum_{z \in \mathbb{C}} w_z(x) = 1$. Moreover, the normalized fuzzy c-means that we introduced in section 4.3, also, provides good results. Both of these variants can be reduced to an expectation over a posterior.

Obviously, this needs more investigation, and a better understanding of the commitment loss hyperparameter α.

5.1.3 Replacing the Quantizer With an Expectation

In [25], they replaced the quantizer $\beta(x)$ with the empirical expectation $\mathbb{E}_{q_{\mathbf{w}}(z|x)}\mathbf{w}_z$ in the reconstruction error. Note that we do not evaluate the expectation, we only sample m fixed number of samples and calculate the emperical average

The number of samples, m, should be a small number. We experimented with one up to three without problems. The reconstruction loss becomes

$$\log p\left(\mathbf{x} \mid \mathrm{z}_\theta(x) - \mathrm{sg}\left[\mathbb{E}_{q_{\mathbf{w}}(z|x)}\left[w_z - \mathrm{z}_\theta(x)\right]\right]\right), \quad z_0, z_1, \ldots, z_m \sim q_{\mathbf{w}}(z \mid x)$$

where the expectation is approximated as follows

$$\mathbb{E}_{q_{\mathbf{w}}(z|x)}\mathbf{w}_z \approx \frac{1}{m}\sum_{i=0}^{m-1}\mathbf{w}_{z_i} \quad \text{such that} \quad z_0, z_1, \ldots, z_m \sim q_{\mathbf{w}}(z \mid x).$$

They claimed that this provides a more stable training. Note that this is the expectation is the one we are trying to minimize in the commitment loss and the VQ loss.

5.2 VQ-VAE as a VAE

Quoting [11], "Despite the name, VQ-VAEs are neither stochastic, nor variational, but they are deterministic autoencoders." So far, we have described VQ-VAE as a regularized autoencoder, but as we have seen in section 5.1.3, VQ-VAE can be trained as in stochastic manner.

Many published research view it as a VAE. In fact, VQ-VAE consists of the components needed for a VAE. The posterior $q_{\mathbf{w}}(z \mid \mathrm{z}_\theta(x))$ is the approximate posterior and $p_\phi(x \mid z)$ is the decoder. The prior can be assumed to be uniform $p(z)$ during training [29].

The closest justification we found for a VQ-VAE to be a VAE is in [30] where they generalized the vanilla VQ-VAE with what they called Hierarchical Quantized Autoencoder (HQA), which is a VAE with GMM prior. Their loss is

$$\ell(\mathbf{x}) = \log p(\mathbf{x} \mid \beta(\mathbf{x})) - \mathscr{H}(q_{\mathbf{w}}(z \mid \mathrm{z}_\theta(\mathbf{x}))) + \mathbb{E}_{q_{\mathbf{w}}(z|\mathrm{z}_\theta(\mathbf{x}))}\|\mathrm{z}_\theta(\mathbf{x}) - (\mathbf{w}_z)\|^2 \tag{5.8}$$

Where $\mathscr{H}(q_{\mathbf{w}}(z \mid \mathrm{z}_\theta(x)))$ is the entropy of the posterior which is the main difference between the loss in 5.8. This loss is similar to 5.8 for the neural gas inspired posterior because the entropy of the neural gas posterior is constant. The case is also the same for the k-means posterior. However, they still needed to introduce a separate commit-

ment loss ($\|\mathrm{z}_\theta(x) - \mathbf{w}_{\gamma(x)\|^2}$ from the vanilla VQ-VAE which does not show up in their derivation of the loss. Moreover, their derivation of the loss contains a lot of approximations.

5.3 Stage 2: Learning the Prior

After we optimized the VQ-VAE loss function, the mapping $\gamma(x)$ can perform compression by mapping the data instance x to the set $\mathbb{C} = \{0, 1, \ldots, k-1\}$. In practice we need huge number of prototypes to be able to encode complex data such as real life images. Thus we apply ideas from section 4.4. Therefore we end up with multiple discrete random variables similar to z that takes values from $\mathbb{C}$, or simply z is a random vector or tensor.

We used CNN to parameterize the encoder and decoder such that the continuous latent space corresponds to a three dimensional tensor that belongs to R^{CxHxW}. One of these dimensions corresponds to the number of channels, while we call H the height and W the width. As we described in section 4.4. Ignoring the channel dimension, the dependency between two elements in the tensor that belong to different width and height dimensions is analogous to the dependency between two pixels in a picture at different height and width. The reason for that is the receptive field of each tensor element is more likely to intersect with close by elements.

As described in section 4.4, we can obtain the following quantization: the prototypes belong to R^C, and our latent z belongs to $\mathbb{C}^{k \times W \times H}$ which correspond to a categorical matrix. This data is both categorical and is analogous to images thus PixelCNN can learn the distribution of z from the set $Vn(X)$, where X is our data set and $Vn(X)$ is the image the set X under the quantizer $Vn(x)$. Basically, You can think of $Vn(X)$ as another data set for PixelCNN. $Vn(X)$ are samples from $p(z)$ and thus we are learning the prior.

Having learned the prior $p(z)$, we can sample from it and pass to the decoder. Thus, generating new data points.

6 Training

The encoder $\mathrm{z}_\theta(\mathbf{x})$ and the decoder $\mathbf{p}_\phi(\mathbf{x} \mid \beta(x))$ are ResNet [13] architectures, please see [13] for details. We use similar architecture to the one in [29] for the neural network part.

The encoder starts by two CNNs of stride 2 and kernel size 4×4. The output of these passed to two residual blocks. Each residual block is a ReLU followed by 3×3 CNN then ReLU and lastly a 1×1 CNN.

The decoder tries to invert the encoder by starting with the same residual blocks in the encoder, followed by two transposed CNNs with stride 2 and kernel 4×4.

The optemizer is Adam [17] with learning rate $1e-3$. We use 512 prototypes, with 64 dimension each. We experimented with the CIFAR10 data set.

We faced some technical difficulty, we lost our small set of test results. Here, we run small tests on the training set instead. For neural gas, we did a quick compression results and they are as follows:

- $\alpha = 0.15,\ \eta = 1,\ \lambda = 3$, reconstruction-error on the training set is 0.058 and perplexity 450 after 30 epochs of training.
- $\alpha = 0.10,\ \eta = 1, \lambda = 3$, reconstruction-error on the training set is 0.06 and perplexity 450 after 20 epochs of training.
- $\alpha = 0.15,\ \eta = 1,\ \lambda = 10$, reconstruction-error on the training set is 0.065 and perplexity 463 after 20 epochs of training.

Note that we divided the model with the batch size on all the above items.

For the posterior in 4.6, a reconstruction loss of 0.62 with $\alpha = 0.08$ and $\eta = 1$ after 20 epochs. However, for fuzzy c-means it diverges on most tests.

7 Conclusion and Future Work

We have discussed different generative models with focus on VQ-VAE. We have shown that VQ-VAE can perform well with neural gas vector quantization and some other stochastic based models. However, it seems that no major improvement can be attained without considering hierarchical models.

Moreover, one major issue is that there is no clear theoretical justification for VQ-VAE and the choice of the hyperparameters. This and the fact that it is very sensitive to the hyper-parameter makes it hard to train as we need to keep the reconstruction error low and the VQ based error low simultaneously. This should be an area to investigate in the future.

Mathematical Symbols

$\mathbb{C}$	The set of cluster indices.
$\mathbb{X}$	The data set.
$\beta(x)$	A vector quantizer, mapping x to a prototype.
$\gamma(x)$	Quantizer encoder, mapping x to $\mathbb{C}$.
$\mathrm{d}(x,y)$	Eucledian distance between x and y.
E	Objective function or energy function.
$\ell(x)$	The loss function on instance x.
$\ell(x)$	The log likelihood function.
$\mathrm{z}_\theta(x)$	The encoder.
$\mathrm{sg}(x)$	The stop gradient operator.
$ELBO$	The evidence lower bound loss function.
D_{KL}	The KL divergence.
$\mathscr{H}(x)$	The entropy of the random variable x .
$\mathscr{H}(x \mid z)$	The conditional entropy of x given z.
$\mathscr{H}(x \mid z)$	The conditional entropy of x given z.
$\odot$	The element-wise product.
$*$	The convolution.
ANN	Artificial Neural Networks.
GMM	Gaussian mixture model.
MLP	Multilayer perceptron.
CNN	Convolutional neural network.
VQ	Vector quantization.
AR	Autoregressive Model.
VAE	Variational autoencoder.
$VQ-VAE$	Vector Quantised-Variational AutoEncoder.

Bibliography

[1] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[2] Jake Bouvrie. Notes on convolutional neural networks. November 2006.

[3] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. The concrete distribution: A continuous relaxation of discrete random variables. In *Proceedings International Conference on Learning Representations 2017*. OpenReviews.net, April 2017.

[4] Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *ICLR (Poster)*, 2016.

[5] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder, 2017.

[6] Danilo Costarelli and Renato Spigler. How sharp is the jensen inequality? *Journal of Inequalities and Applications*, 2015(1):69, Feb 2015.

[7] Marie Cottrell, Barbara Hammer, Alexander Hasenfuß, and Thomas Villmann. Batch and median neural gas. *Neural Networks*, 19(6):762–771, 2006. Advances in Self Organising Maps - WSOM'05.

[8] M. Montserrat Alonso Ferrero. Voronoi diagram: The generator recognition problem, 2011.

[9] Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. Som-vae: Interpretable discrete representation learning on time series, 2019.

[10] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR.

[11] Partha Ghosh, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Scholkopf. From variational to deterministic autoencoders. In *International Conference on Learning Representations*, 2020.

[12] R. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[14] Chin-Wei Huang, Kris Sankaran, Eeshan Dhekane, Alexandre Lacoste, and Aaron Courville. Hierarchical importance weighted autoencoders. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2869–2878. PMLR, 09–15 Jun 2019.

[15] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumbel-softmax. In *Proceedings International Conference on Learning Representations 2017*. OpenReviews.net, April 2017.

[16] Jaeyoon Kim, Donghyun Tae, and Junhee Seok. A survey of missing data imputation using generative adversarial networks. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 454–456, 2020.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[18] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 29–37, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[19] David J. C. MacKay. *Information Theory, Inference Learning Algorithms*, pages 284–289. Cambridge University Press, USA, 2002.

[20] David J. C. MacKay. *Information Theory, Inference Learning Algorithms*, pages 138–140. Cambridge University Press, USA, 2002.

[21] T.M. Martinetz, S.G. Berkovich, and K.J. Schulten. 'neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993.

[22] D. Nebel, M. Kaden, A. Villmann, and T. Villmann. Types of (dis-)similarities and adaptive mixtures thereof for improved classification learning. *Neurocomputing*, 268:42–54, 2017. Advances in artificial neural networks, machine learning and computational intelligence.

[23] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural

networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1747–1756, New York, New York, USA, 20–22 Jun 2016. PMLR.

[24] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[25] Aurko Roy, Ashish Vaswani, Arvind Neelakantan, and Niki Parmar. Theory and experiments on vector quantized autoencoders, 2018.

[26] Hoo-Chang Shin, Neil A. Tenenholtz, Jameson K. Rogers, Christopher G. Schwarz, Matthew L. Senjem, Jeffrey L. Gunter, Katherine P. Andriole, and Mark Michalski. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In Ali Gooya, Orcun Goksel, Ipek Oguz, and Ninon Burgos, editors, *Simulation and Synthesis in Medical Imaging*, pages 1–11, Cham, 2018. Springer International Publishing.

[27] Hoo-Chang Shin, Neil A. Tenenholtz, Jameson K. Rogers, Christopher G. Schwarz, Matthew L. Senjem, Jeffrey L. Gunter, Katherine P. Andriole, and Mark Michalski. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In Ali Gooya, Orcun Goksel, Ipek Oguz, and Ninon Burgos, editors, *Simulation and Synthesis in Medical Imaging*, pages 1–11, Cham, 2018. Springer International Publishing.

[28] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, koray kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[29] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[30] Will Williams, Sam Ringer, Tom Ash, David MacLeod, Jamie Dougherty, and John Hughes. Hierarchical quantized autoencoders. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4524–4535. Curran Associates, Inc., 2020.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, im May 2021