Exploring Variational Autoencoders (VAEs) for deep learning

Youpeng Zhao, Tianyu Zhan, Wenqi Shi and Rao Fu

Georgia Institute of Technology

Abstract

In the last few years, deep learning based generative models have gained more and more interest due to (and implying) some amazing improvements in the field. Relying on huge amount of data, well-designed networks architectures and smart training techniques, deep generative models have shown an incredible ability to produce highly realistic pieces of content of various kind, such as images, texts and sounds. One of the major families that stands out are Variational Autoencoders (VAEs). In this project, we explored the working mechanism for VAEs and how they are used in vision tasks.

1. Introduction

Recent advances in Deep Learning techniques have encouraged a new trend in Machine Learning research, especially in the area of unsupervised learning. The opportunity of having unlimited data at hand, which can be used to improve current machine learning solutions is tempting yet challenging. In particular, Variational Auto-Encoder(VAE) has been one of the most popular and promising approaches for unsupervised learning tasks.

The focus of this project is to implement the vanilla VAEs from scratch, based on the original paper[4], and explore its variations in performing vision tasks, such as reconstruction of images, and generation of unseen images. In this report, Section **2** introduces the architecture and working mechanism of VAEs. Section **3** includes the implementation of VAEs in Tensorflow. Section **4** concludes the report and discusses future work.

2. Variational Autoencoders

In Bayesian machine learning, the variational inference is often required to achieve the optimization of an approximation to the intractable posterior. Variational autoencoder (VAE) is a directed generative model with both observed and latent variables, and has merged as a popular unsupervised learning method. It models a data distribution using a collection of independent latent variables. In general, VAEs are powerful estimator of variational lower bound for efficient approximate inference with continuous latent variables and can be straightforwardly differentiated and optimized using standard stochastic gradient methods.

2.1. Problem Statement

2.1.1 Auto-Encoders (AEs)

Auto-encoder is a neural network used to learn encoding data in an unsupervised manner. It consists of two parts: an encoder that encodes high-dimensional data into low-dimensional latent representation and a decoder that decodes the latent information to generate outputs. The entire network is usually trained as a whole, where the loss function, also known as reconstruction loss, penalizes the network for creating outputs different from the input. The encoder learns to preserve as much of the relevant information as possible in the limited encoding, and intelligently discard irrelevant parts while the decoder learns to take the encoding and properly reconstruct it into a full image.

Standard autoencoders learn to generate compact representations and reconstruct corresponding inputs well. However, the fundamental problem with autoencoders is discontinuities: the latent space of AEs convert their original inputs to and where the encoded vectors lie may not be continuous or allow easy interpolation.

2.1.2 Problem scenario

We aim at deriving a lower bound estimator for a variety of directed graphical models with continuous latent variables. Consider a dataset $X = \{x^{(i)}\}_{i=1}^{N}$, consisting of N independent and identically distributed (i.i.d) samples of some unknown variable x. Assume that a value $z^{(i)}$ is generated from some prior distribution $p_{\theta^*}(z)$ and the value $x^{(i)}$ is generated from some conditional distribution $p_{\theta^*}(x|z)$. In our assumption, the prior $p_{\theta^*}(z)$ and likelihood $p_{\theta^*}(x|z)$ come from distribution $p_\theta(z)$ and $p_\theta(x|z)$ while the true parameters θ^* and the values of the latent variables $z^{(i)}$ are hidden from our view. In addition, we assume that computing the posterior probability $p_\theta(x|z)$ is intractable and the dataset X is too large to fit in memory as we can only work with small, subsampled batches instead of the whole dataset. VAE is introduced to solve the related problems in the above scenario:

1. Efficient approximate maximum likelihood (ML) or maximum a posteriori (MAP) estimation for the parameters θ.
2. Efficient approximate posterior inference of the latent variable z.
3. Efficient approximate marginal inference of the variable x.

For the purpose of solving the problems above, we introduce $q_\phi(z|x)$ as a probabilistic encoder: with a given datapoint x it produces a distribution over the possible values of the code z from the real distribution. On the other hand, the likelihood $p_\theta(x|z)$ can be viewed as a probabilistic decoder: with a given code z, it can generate a distribution over the possible corresponding values of x.

2.2. Variational Inference

The main idea of variational methods is to cast inference as an optimization problem[2], utilized to approximate a conditional density of latent variables given some observed variables. The basic idea behind variational inference is to posit a family of densities and find the member of that family which is close to the target. More specifically, variational techniques try to solve an optimization problem over a class of tractable distributions Q in order to find a $q \in Q$ that is most similar to p. Variational inference has been widely utilized in different applications and tends to be faster than classical methods, such as Markov chain Monte Carlo sampling. In Bayesian machine learning, we usually use Kullback-Leibler (KL) Divergence to capture the similarity between q and p and perform variational inference with a KL divergence.

2.3. Kullback-Leibler Divergence

Let $p(x)$ and $q(x)$ are two probability distributions of a discrete random variable x. The Kullback-Leibler divergence, which is closely related to relative entropy, information divergence, and information for discrimination, is a non-symmetric measure of the difference between two probability distributions $p(x)$ and $q(x)$. Typically, $p(x)$ represents the real distribution of data, observations, or a precisely calculated theoretical distribution. The measure q(x) typically represents a theory, model, description, or approximation of p(x).

In general, the higher the probability of an event, the lower its information content. The KL divergence measures the expected number of extra bits required to code samples from $p(x)$ when using a code based on $q(x)$, rather than using a code based on $p(x)$ itself. First, we can model the information content of event x:

$$\begin{aligned} p = I_p(x) &= -\log p(x) \\ q = I_q(x) &= -\log q(x). \end{aligned} \tag{1}$$

Then we can measure the difference between $q(x)$ and $p(x)$:

$$\Delta I = I_p - I_q = -\log p(x) + \log q(x) = \log\left(\frac{q(x)}{p(x)}\right). \tag{2}$$

By definition, the KL divergence is the expectation of the above difference and can be represented as:

$$\begin{aligned} D_{KL}(q(x)\|p(x)) &:= E_{\sim q}[\Delta I] \\ &= \int (\Delta I) q(x) dx \\ &= \int q(x) \log\left(\frac{q(x)}{p(x)}\right) dx. \end{aligned} \tag{3}$$

Although the KL divergence measures the "distance" between two distributions, it is not a distance measure because of the non-symmetry.

$$D_{KL}(q(x)\|p(x)) \neq D_{KL}(p(x)\|q(x)). \tag{4}$$

2.4. The evidence lower bound (ELBO)

In variational inference, we specify a family Q of densities over the latent variables. Let $\mathbf{x} = x_{1:n}$ be a set of observed variables and $\mathbf{z} = z_{1:m}$ be a set of latent variables, with joint density $p(\mathbf{z}, \mathbf{x})$. Each $q(\mathbf{z}) \in Q$ is a candidate approximation to the exact conditional. As an optimization problem, we need to specify the closest candidate in KL divergence to the exact conditional. In practice, instead of minimizing the KL divergence exactly, we can minimize a function equal to it up to a constant, the evidence lower bound (ELBO). Inference now amounts to solving the following optimization problem,

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in\mathcal{Q}}{\arg\min}\, \mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})) \tag{5}$$

Once found, $q^*(\cdot)$ is the best approximation of the conditional, within the family Q and the complexity of the family determines the complexity of this optimization. However, this objective cannot be directly obtained so that we expand the conditional of KL divergence,

$$\mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})) = \mathbb{E}[\log q(\mathbf{z})] - \mathbb{E}[\log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x}). \tag{6}$$

Then we can optimize an alternative objective equivalent to the KL up to an added constant and get the evidence lower bound (ELBO).

$$\mathrm{ELBO}(q) = \mathbb{E}[\log p(\mathbf{z}, \mathbf{x})] - \mathbb{E}[\log q(\mathbf{z})]. \tag{7}$$

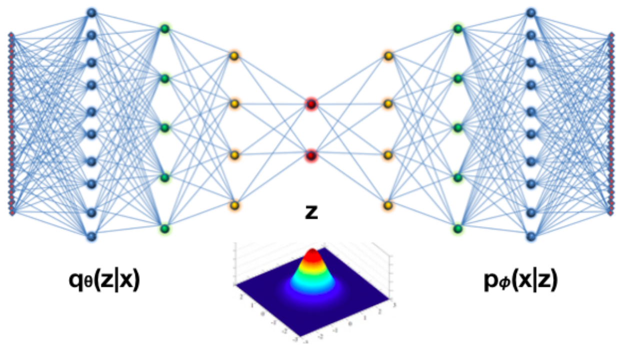


Figure 1. The sketch of network structure in variational autoencoders.

The ELBO is negative KL divergence plus $\log p(x)$, i.e. maximizing the ELBO is equivalent to minimizing the KL divergence. We rewrite the ELBO as a sum of the expected log likelihood of the data and the KL divergence between the prior $p(z)$ and $q(z)$,

$$\begin{aligned}\mathrm{ELBO}(q) &= \mathbb{E}[\log p(\mathbf{z})] + \mathbb{E}[\log p(\mathbf{x}|\mathbf{z})] - \mathbb{E}[\log q(\mathbf{z})] \\ &= \mathbb{E}[\log p(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z}))\end{aligned} \tag{8}$$

In the ELBO representation, the first term is an expected likelihood leading densities that place their mass on configurations of the latent variables; the second term, the negative divergence between the variational density and the prior, encourages densities close to the prior. Therefore, the variational objective mirrors the usual balance between likelihood and prior.

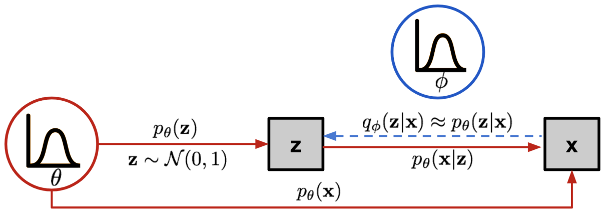
2.5. Variational Autoencoders (VAEs)

The idea of Variational Autoencoders (VAEs) differs significantly from classical in terms of the formulation of mathematics. VAEs are generative models, which aims to learn the generative process that can be used to generate random instances. VAEs are Directed Probabilistic Graphical Models (DPGM)[3], whose posteriror is approximated by a neural network, forming an auto-encoder.

In VAEs, instead of mapping the input data to a fixed vector in latent spaces, the goal is to map it into a distribution p_θ, parameterized on a neural network by weights collectively denoted θ. The generative process can be defined by:

$$p_\theta(\mathbf{x}) = \int_z p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}. \tag{9}$$

The encoder of a VAE, $q_\theta(z|x)$, yields an approximate posterior distribution $q(z|x)$ parameterized by θ. The decoder of VAE, $p_\phi(x|z)$, yields a likelihood distribution $p(x|z)$ and parameterized on a neural network by weights collectively denoted ϕ. The output of the encoder are parameters of the latent distribution, which is sampled to yield the input into the decoder. Figure 2 shows the graphic model involved in VAEs.



θ
ϕ
$p_\theta(\mathbf{z})$
$\mathbf{z} \sim \mathcal{N}(0,1)$
z
$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$
$p_\theta(\mathbf{x}|\mathbf{z})$
x
$p_\theta(\mathbf{x})$

Figure 2. The graphical model involved in Variational Autoencoder. Solid lines denote the generative distribution $p_\theta()$ and dashed lines denote the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$.[7]

The goal of VAEs contains two parts: one is to maximize the log-likelihood of data, and another is to get estimated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ as close to $p_\theta(\mathbf{z}|\mathbf{x})$ as possible. We use the KL divergence to measure the difference between two distributions:

$$\begin{aligned}&D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})\,|p_\theta(\mathbf{z}|\mathbf{x})\right) \\ &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\ &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \\ &= \int q_\phi(\mathbf{z}|\mathbf{x}) \left(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})}\right) d\mathbf{z} \\ &= \log p_\theta(\mathbf{x}) + D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})|p_\theta(\mathbf{z})\right) - E_{\mathbf{z}\sim q_\phi(\mathbf{x}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}))\end{aligned} \tag{10}$$

After rearranging the left and right hand side of the equation, we can get the loss function for VAEs L_{VAE} in the form of ELBO,

$$L_{VAE}(\theta,\phi) = -\log\mathrm{p}_\theta(\mathbf{x}) + D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})\,|p_\theta(\mathbf{z}|\mathbf{x})\right) \tag{11}$$

As for training the model parameters θ and ϕ, instead of directly sampling from z, a stochastic process that cannot back-propagate the gradient, a Gaussian variable ϵ is sampled from $\mathcal{N}(0,I)$, where$z = \mu + \sigma \odot \epsilon$. In order to make the network differentiable, we implement reparameterization trick[5] to divert the non-differentiable operation out of the network. Intuitively, in its original form, VAEs sample from a random node z which is approximated by the parametric model $q(z|\phi,x)$ of the true posterior.

However, back-propagation cannot flow through a random node, so that we introduce the new parameter ϵ, allowing us to reparameterize z in a way that allows back-propagate to flow through the deterministic nodes. We can figure out the difference between original form and reparameterization trick in Figure 3. This particular trick allows the mean and log-variance vectors to still remain as the learnable parameters of the network while still maintaining

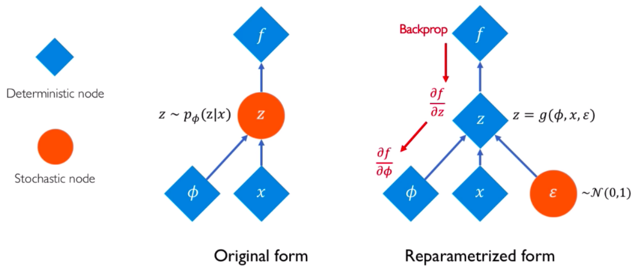


Figure 3. Illustration of variational autoencoder network with and without the reparameterization trick[1].

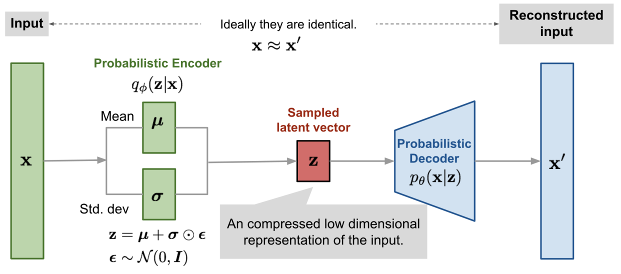

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
Std. dev
Sampled latent vector
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
An compressed low dimensional representation of the input.
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$

Figure 4. Illustration of variational autoencoder model with the multivariate Gaussian assumption[7].

the stochasticity of the entire system via ϵ. For illustration, Figure 4 gives a detailed example of illustration for VAEs with multivariate Gaussian assumption.

3. Implementation

In this Section, we implemented Variational Autoencoders in Tensorflow. To simplify our code implementation, we used Keras, a high-level neural networks API, running on top of Tensorflow. We first implemented vanilla VAEs, consisting of Fully Connected (FC) layers only; then, we introduced additional layers into the network, such as convolutional layers, dropout layers, to compare the performance with vanilla VAEs.

3.1. Dataset

Due to the complexity of our model and our computational limit, we feel that MNIST, a large database of handwritten digits, is rather appropriate for our experiments.

3.2. Vanilla VAEs

In vanilla VAES, we choose three hidden layers, with middle one being the latent space z. The dimension of encoder hidden layer is 256, same as decoder hidden layer. We first set latent space dimension $z = 2$, which enables us to visual latent space representation in a 2D plot.

We can see in Figure 5, in latent space, different digits represents different clusters. This means that VAEs actually learned which distribution corresponds to which handwritten digit. This proves that our model has indeed learned the underlying structure of our data.

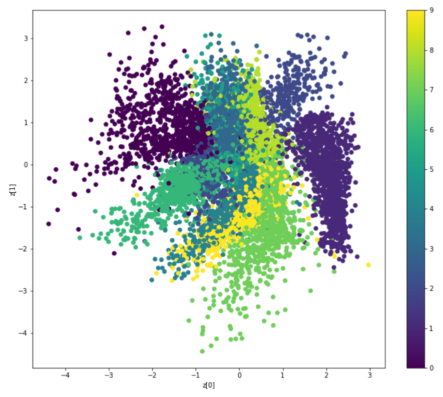


Figure 5. Latent space representation in 2D

For our training parameters, each epoch, we use 6000 samples for training and 1000 samples for validation. Batch size is 64, training epochs is 50. We use ReLU and Sigmoid as our activation functions.

We tested the model on different size of latent space dimension ($z = 2, 5, 16$), and compared the final loss, total number of parameters and FID[6] score between the original images and reconstructed images. FID score is mostly used for measuring GAN performance, but here is used to measure the image quality. Lower FID scores mean better image quality and diversity.

In Table 1, we can see that, as z gets larger, our training and validation loss gets smaller, and FID score gets lower, which indicates that the image quality is getting better. This can also be seen by qualitative analysis of our reconstructed images.

In Figure 6 and 7 (each column represents a sequence of images, columns of odd numbers are original images, and the rest are reconstructed images), we can see that $z = 16$ has much better image reconstruction against $z = 2$.

Apart from image reconstruction, another task we could perform is generating unseen images. For this task, we sample random z in latent space, and feed into our trained decoder network to generate images. Since z is sampled randomly, and we only trained limited samples, the generated images will be mostly unseen. Here, for demonstration purpose, we still choose $z = 2$. We can see the generated MNIST manifold in Figure 8.

3.3. Convolutional VAEs

In vanilla VAEs, we only use FC layers in the network. The main drawback of "fully-connectedness" is the problem of overfitting. Here, we introduce the convolutional layers

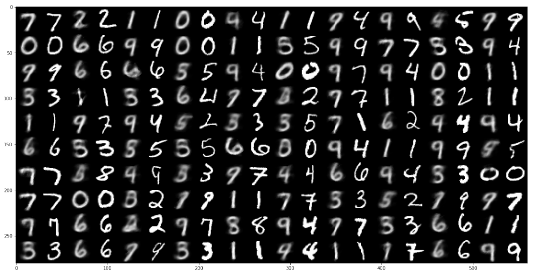


Figure 6. Image reconstruction ($z = 2$).

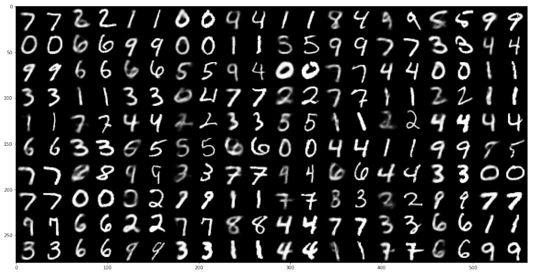


Figure 7. Image reconstruction ($z = 16$).

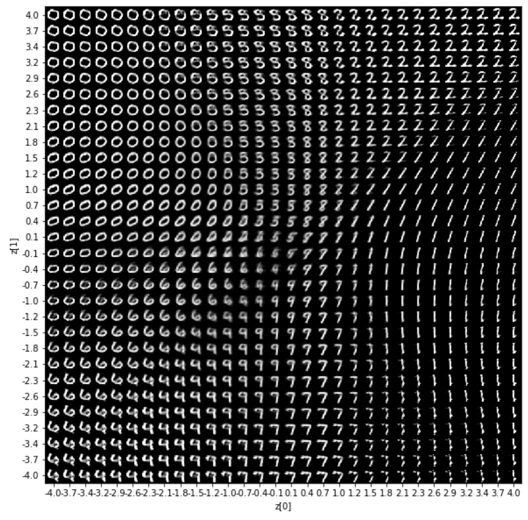


Figure 8. Generated MNIST manifold in 2D.

Table 1. Performance of vanilla VAEs with different $z = 2, 5, 16$.

	Train Loss	Val Loss	FID	Parameters
$z = 2$	37.49	37.90	19.93	404,244
$z = 5$	31.09	31.14	8.67	406,554
$z = 16$	29.43	29.24	4.63	415,024

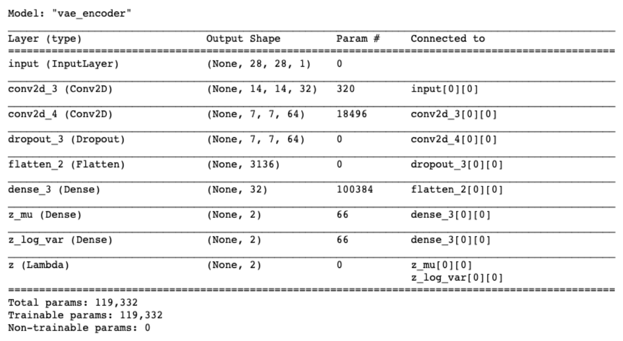
and droptout layers to solve this problem.

Convolutional layers are the core building blocks of CNN. Parameters consist of a set of trainable filters, which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2D activation map of that filter. Therefore, for each specific dataset, choosing right size of filter and channel is essential.

Dropout layers are also used for reducing overfitting. At each training stage, individual nodes are either "dropped out" of the net with probability $1-p$ or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Dropout essentially forces the model to learn more robust features to better generalize to new data.

For MNIST dataset, here, we choose two convolutional layers and one dropout layer in our encoder network; two deconvolutional layers and one dropout layer in our decoder network. Using Keras, we can implicitly list out our encoder/decoder network structures. Figure 9 and 10 are our encoder and decoder network summary.

For our training parameters, same as before. In convolutional layers, we use 3 by 3 filter, with 32 and 64 channels, two strides with zero paddings. Dropout probability is 0.25. We also tested the model on different size of latent space dimension ($z = 2, 5, 16$), and furthermore, compared with vanilla VAEs.


```
Model: "vae_encoder"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              (None, 28, 28, 1)    0
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 14, 14, 32)   320         input[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 7, 7, 64)     18496       conv2d_3[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 7, 7, 64)     0           conv2d_4[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 3136)         0           dropout_3[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 32)           100384      flatten_2[0][0]
__________________________________________________________________________________________________
z_mu (Dense)                    (None, 2)            66          dense_3[0][0]
__________________________________________________________________________________________________
z_log_var (Dense)               (None, 2)            66          dense_3[0][0]
__________________________________________________________________________________________________
z (Lambda)                      (None, 2)            0           z_mu[0][0]
                                                                 z_log_var[0][0]
==================================================================================================
Total params: 119,332
Trainable params: 119,332
Non-trainable params: 0
```
Figure 9. Convolutional VAE encoder network.

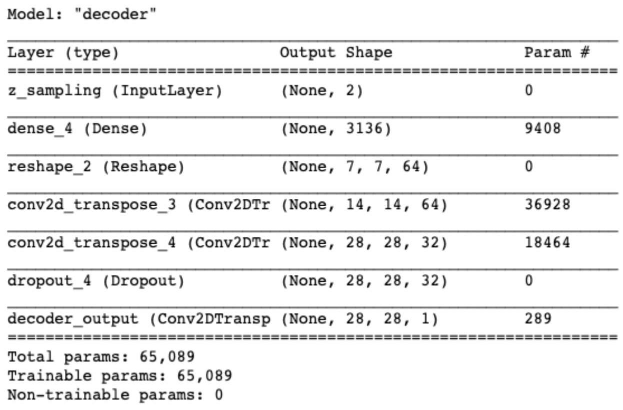

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      (None, 2)                 0
_________________________________________________________________
dense_4 (Dense)              (None, 3136)              9408
_________________________________________________________________
reshape_2 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
dropout_4 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
decoder_output (Conv2DTransp (None, 28, 28, 1)         289
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
```
Figure 10. Convolutional VAE decoder network.

Table 2. Performance of Convolutional VAEs with different $z = 2, 5, 16$.

	Train Loss	Val Loss	FID	Parameters
$z = 2$	36.72	36.62	17.20	184,421
$z = 5$	30.27	29.77	7.12	194,027
$z = 16$	33.19	32.59	10.59	229.249

In Table 2, we can see that Convolutional VAEs have lower FID score when $z = 2$ and $z = 5$. We see an increase of FID score and loss when $z = 16$. We assume it might have something to do with choosing the right dimension of latent space z. But to compare with vanilla VAEs, our model has much less parameters to train and theoretically more robust.

4. Conclusion

In this project, we explored VAE, one of the most popular deep learning framework for unsupervised learning. We first looked into the theoretical part of VAE, understood the basic framework of VAE, and how it worked. Next, we implemented VAE in Tensorflow for MNIST: we first built the vanilla VAE, and then added convolutional and dropout layers to further improve the complexity and robustness of the model. We examined the latent space representation to see if VAEs actually learn the underlying structure of the dataset and used loss and FID score to compare the performance. We see that choosing the right latent space dimension is very critical to our model, when optimizing the performance.

For future work, we want to further explore how convolutional layers affect the performance of the model, such as filter size, stride, padding. Also, we would like to add additional pooling layer to improve robustness and generalization. Last but not least, we want to extend our model to a larger dataset of real-world images for more practical use.

References

[1] Alexander Amini and Ava Soleimany. MIT Introduction to Deep Learning. `http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L4.pdf`, 2020.

[2] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[3] Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. pages 2946–2954. Curran Associates, Inc., 2016.

[4] Diederik Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[5] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019.

[6] Thomas Unterthiner Martin Heusel, Hubert Ramsauer and Bernhard Nessler. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NIPS*, 2017.

[7] Liliang Weng. From Autoencoder to Beta-VAE. `https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html`, 2018. Online.

5. Contribution

Tianyu Zhan and Wenqi Shi, mostly worked on deriving the theoretical part of VAE and analyzed the results.

Youpeng Zhao worked on implementing and training the VAE network, while Rao Fu was involved in parameter tuning and visualization of results.