IMAGE ANALYSIS (SSY098)
Project 4: Generative Neural Networks

Hariharan Gopinath

Systems, Control and Mechatronics
Chalmers University of Technology
hargop@student.chalmers.se

Abstract

The Variational Autoencoder (VAE) is the extended part of the traditional autoencoder and this methods are mostly used in deep generative network models capable of learning unsupervised latent representations of data. We designing a VAE-based generative model which is capable of extracting features correlated to binary labels in the data and structuring it in a latent subspace which is easy to interpret. Here, we implement this model to generate images of digits between 0 and 9.

1. Introduction

Auto encoding are a type of Artifical neural networks that are used to perform a task of data encoding (representation learning) The varitaional autoencoder(VAE) is a mostly widely and rapidly developing technology in the field of deep learning and computer vision. This method is the extension of the traditional autoencoder method except implementing the reparameterization trick in the VAE. Also, this method is the base of the GAN method which is the advancement of the VAE method. The import parts of the VAE is the encoder and the decoder part. As seen in the figure 1, In between the encoder and the decorder, there is a latent state representations of the input data. This latent space is the output of the encoder part and also the input for the decorder part.[1]

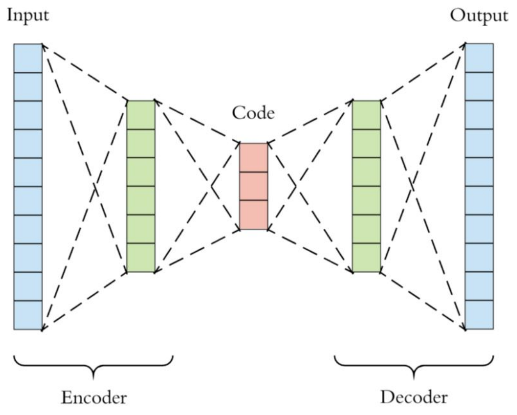
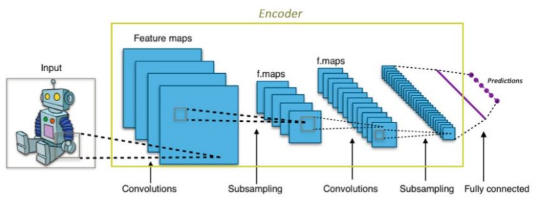


Figure 1: *Encoder and decoder*
Photocredits https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

2. Method

2.1. Autoencoder vs VAE

Auto encoders work well by adding a bottle neck in the network. The bottleneck is nothing but a combination of the various vectors and that forces the network to create a compressed (encodered) version of the original input. Also, it's works well if the correlation exists between the input data (performs poorly if the all input data is independent) A VAE mostly works by implementing a probability distribution such as gaussian distributions and learns from that distribution then finally sampling is done to generate the data-set. A major issue with the regular auto encoder is that the latent space that the input are converted to are discrete (not continuous) and does not allow for an easy interpolation. The generative part of the auto encoder works by randomly picking the samples from the latent space which is challenging if it's discontinuous or has gaps. Variational autoencoders are here to solve this issue.



2.2. Working principle

As I already said, the VAE has two important network, Encoder and the decoder part. The encoder part consist of the convoltional neutral network layers with the relu and finally a fully connected part at the end. The decoder will take that informations from the latent space and try to decodes the output from encoder and generates the original image. using the other type of convoltional layer called transposed convoltional 2D Layer. So that the output will be the imforation taken from the latent space

2.3. Statisical motivation

A Variational Auto-Encoder (VAE) mainly works by implementing a probability distribution (gaussian distribution) and learning from that distribution, then finally sampling is done to generate the learned data-set. The variational autoencoders have a contineous latent space by default which makes them powerful in generating new images.In VAEs,the encoder does not generate a vector of size n but it generate two vectors instead as follows: 1) vector mean μ and the standard deviations σ. Then the decoder can start sampling from this distribution.

2.4. Loss function

The loss function in the neural network nothing but the minimizing the error in the network. In general, The training means minimizing the loss functions but in variational interference, we try to minimize the **ELBO** function. This acts as a optimizer in the neural network.
Let's consider the encoder distribution be like $p(x|z)$ and the decoder distribution $q(z|x)$ as shown in figure 3. Introducing a term called KL divergence which is the measure between the two probability distribution. So, if we wants to ensure that the two distributions are like same, we would like to minimize the KL divergence between the two distributions.

$$ELBO = E_{q(z|x)} logp(x|z) - KL(q(z|x)||p(Z)) \quad (1)$$

Where $q(z|x)$ is the encoder part
$p(x|z)$ is the decoder part
KL is the Kullback-Leibler divergence

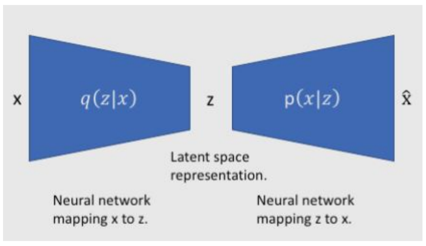

x
q(z|x)
z
p(x|z)
x̂
Latent space representation.
Neural network mapping x to z.
Neural network mapping z to x.

Figure 3: *Distributions*
Photocredits https://www.jeremyjordan.me/variational-autoencoders/

The above function is the loss function(ELBO) and the first term represent the reconstruction likelihood which can be thought of maximizing the reconstruction image likelihood and the second term represent that the learned distribution from the decoder to be similar to the true prior distribution p(z) where these two distribution follow the gaussian distribution KL divergence expression can be written as

$$KL = -\beta \sum_{i=1}^{n} (1 + log(\sigma) - {\mu_i}^2 - \sigma_i^2) \quad (2)$$

Where β-weight to the KL divergence term with a parameter

2.5. Sampling

Before diving into the reparameterization trick, we need to get some clarification about the sampling technique in the our distribution. Let's go the encoder and the decoder part. During the training phase we will be feeding the image input and make the model to learn the encoder and the decoder parameters required to reconstruct the image again so during the testing time, We only need the decoder part because this is the part that generates the image. To do this, we need to input some vector however we have no idea about the nature of the vector if we give some random values more likely than not we will end up with an image unrealisitic or like more noisy garbage image. So we need some method to determine the hidden vector. So, we sue the method called **sampling from the distribution**. This distribution is kind of pool. When we pass the digit images through the encoder part, we will create a random discrete pool of vectors for each digits. Now, Sampling is randoming picking a vector pool from the randomly distributed distribution. During the testing part we may have chance to pick a wrong pool of vector or some non-realistic images. For this reason we using a technique called reparameterization trick.

2.6. Reparameterization trick

One of important features of the VAE is this reparameterization trick. Here, we first wants to constrain the distribution that constrained the region from where we wants to pick the vectors and within this region, the goal of the VAE is to find the pools of the different digits and this is done during the training phase. During the testing phase, we need to generate an image that is randomly sample a vector from this known region and then pass this vector to a Decoder part of the VAE. This will generate an image of the digits. The important property of this method is these are in continuous region. So, we can alter the values to get different looking images (different images). So for the 0-9 digits, we create a constrained pool and try to randomly sampling from this pool to get different images that get changed sightly each other as shown in the figure(4).

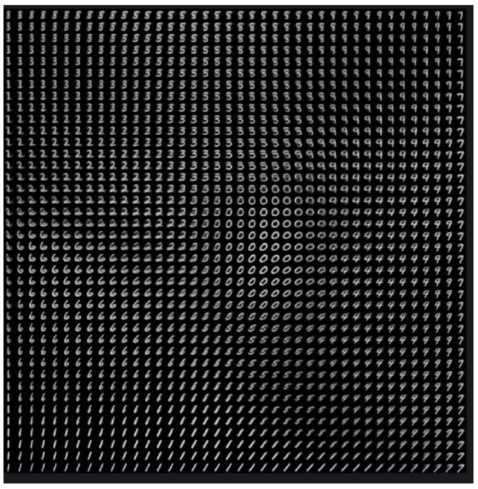


Figure 4: *Sampling pool output*
Photocredits https://www.jeremyjordan.me/variational-autoencoders//

Now, We can discuss the technical aspects of the reparameterization trick. The above sampling process requires some extra works. when we start training our model, we need to define the relationship between the model(output images) and the data. This means that we need to find the parameters that is responsible for classifying the each correct digits image. For finding these parameters, we use the Back-propagation techniques during the training phase. To excecute this technique, our model should be in continuous random distribution. So, we are now randomly sampling ε(0,1) from a unit Gaussian, and then shift the randomly sampled by the latent distribution's mean μ and scale it by the latent distribution's variance σ as shown in the figure 5.

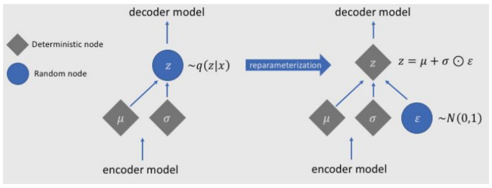


Figure 5: *reparameterization trick*
Photocredits https://www.jeremyjordan.me/variational-autoencoders//

After using this trick, we can execute the back propagation technique because this one requires the chain rule to be applied to the networks and finding the suitable parameters for the systems. The chain rule application can be shown in the figure 6

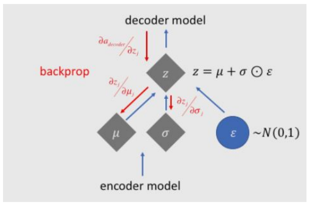


Figure 6: *back propagation*
Photocredits https://www.jeremyjordan.me/variational-autoencoders//

2.7. Optimization

Optimization is the one of the important technique in the field of deep learning. During the training phase network, We initially implement the stochastic gradient descent using back propagation algorithm to determine the gradients. This method helps a lot in achieving the low cost function for the network. This Optimization can be performed with various version of methods but here we will be using the one of the efficient way of Optimization called ADAM which is derived from adaptive moment estimation. Before diving into that, we can explain about the basic gradient method. Firstly, we use the batch gradient descent method which utilize the newton method of Optimization. This Optimization can be expressed as

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{3}$$

Where θ-The parameters
η-learning rate
(θ)- The gradient of the parameters. Following this, the developed version of this process is the **Stochastic gradient descent**. In this method, we will compute the gradient for the mini-batch of examples like hundreds or thousands of examples during the training process. Also, we introduce a another word called **Momentum** [5]. Momentum is a method that helps in accelerating Stochastic gradient descent process in the relevant direction and dampens oscillations as can be seen in figure 7 and 8.

Using the value momentum is like pushing the ball from the uphill rather than natural falling, which will accelerate with more momentum acted on the ball. This same way to our parameter

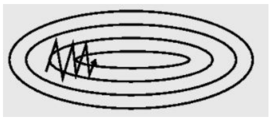


Figure 7: *Without momentum*
Photocredits https://ruder.io/optimizing-gradient-descent/index.htmlfn14

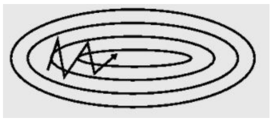


Figure 8: *With momentum*
Photocredits https://ruder.io/optimizing-gradient-descent/index.htmlfn14

updates: The momentum term increases, as a result, we gain faster convergence and reduced oscillation.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Where γ is the momentum factor and ideal value is 0.9
In our VAE, we using the most efficient method of optimization called ADAM is another method that computes adaptive learning rates for each parameter. It's same like SDM with the method but it keeps an exponentially decaying average of past gradients. More details can be seen in this paper for more derivation and calculation.
For our application, We use the MATLAB In-build function for ADAM **adamupdate** [4]. This helps in calculating the Adam optimization algorithm to update network parameters in training loops that use networks defined as dlnetwork objects or model functions.

2.8. Network structure

In encoder structure, I have used two 2D convolutional layer along with the two RELU layers. Like same is used in decoder network but we have been using the deconvolutional layer along with the 2 RELU layers.In between that, we using the 2 dim latent space with two mean and two variance as shown in figure 9.

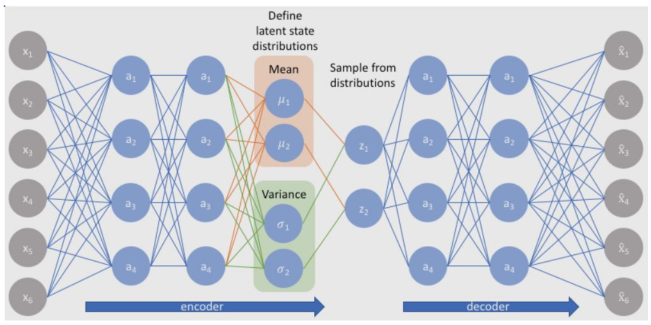

Define latent state distributions
Mean
Variance
Sample from distributions
x_1 x_2 x_3 x_4 x_5 x_6
a_1 a_2 a_3 a_4
μ_1 μ_2
σ_1 σ_2
z_1 z_2
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$ $\hat{x}_5$ $\hat{x}_6$
encoder
decoder

Figure 9: *Network structure 2D latent space*
Photocredits https://www.jeremyjordan.me/variational-autoencoders/

2.9. Network test Evaluation

In this section, we are going to discuss about the different results of this structure analysis and trying to pick out the best efficient one of them. In the Matlab, I have done 12 different test cases of the network to determine the best one out of them. **In the first case**, I used momentum as one, two conv and deconv layers and the learning rate is 0.001 as the ideal one for many test cases. The output seems to be somewhat good but with some images as blurry as seen in the figure 10. This also due to the 2D latent space because the many images are constrained in a small space.

The image from the normal distribution can be shown in figure 10

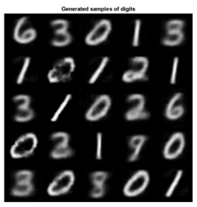


Figure 10: *sampled from a normal distribution*

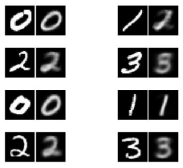


Figure 11: *Network structure 2D latent space*

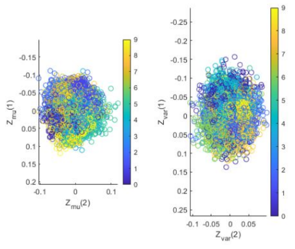


Figure 12: *mean and the variance encoder*

So I have increased the no of convolutional layers on both encoder and decoder part and the results seems to be good when compared to the last one but still some blurriness as seen in the figure 13.
Then, By increasing and decreasing the learning rate as 0.01 or 0.0001, the results are not such good as seen in the figure 14 when comparing with the results from the learning rate 0.001. This makes the layer to be more difficult to identify the output as shown in figure 14.

Then, I analysis the network with a another concept called **dropout** [6] in both the encoder and the decoder section. Dropout is a regularization method used in machine learning that approximates training a large number of neural networks with different architectures in parallel. This dropout method

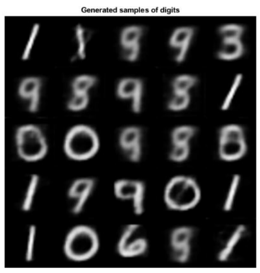


Figure 13: *more conv layers*

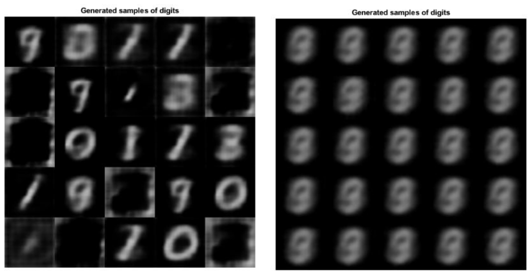


Figure 14: *Change in learning rate*

helps in preventing the model from over fitting and this is done by dropping out or ignoring randomly some number of layer outputs or neurons so that the model neuron will try to learn newly.
In our model, I tried two cases in dropout, Firstly the dropout in the encoder section and dropout in both the encoder and the decoder section. In the first cases, the results are not that much pretty and because this affects the sampling process in the latent space. Then in the second case, I tired with more epocs, the results seems to be good around the centre of the sampling sapce as seen the figure 15-b

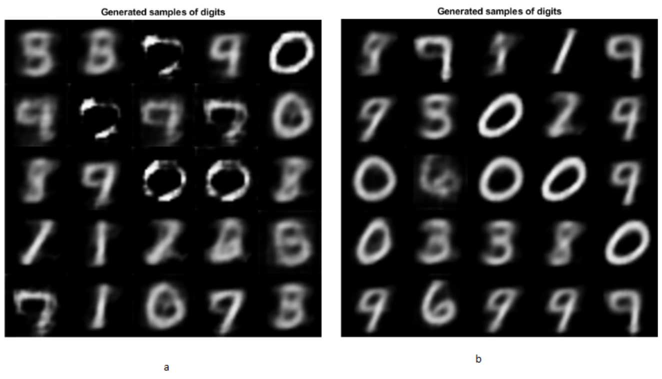


Figure 15: *a-dropout in encoder section,b-dropout in both*

Finally, the good results came out well when used the 5D latent space with optimal learning rate and momentum of 0.001 and 1 respectively. Also, I have increased the no of layers without the dropout layer. This is the final test results with good results as shown in the figure 16.

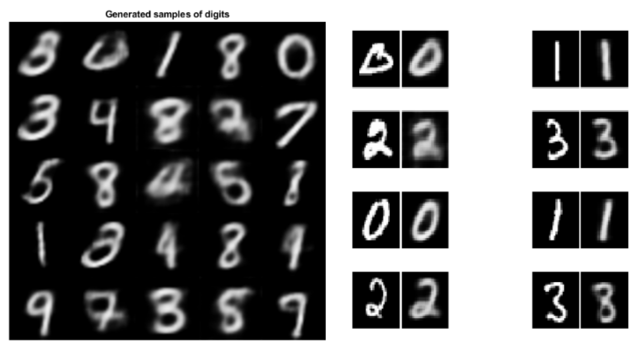


Figure 16: *5D latent space*

3. Theoretical question

1) In your own words, explain why we optimize the Empirical Lower Bound (ELBO) rather than $log(p(x))$ when training a VAE. See the figure 17

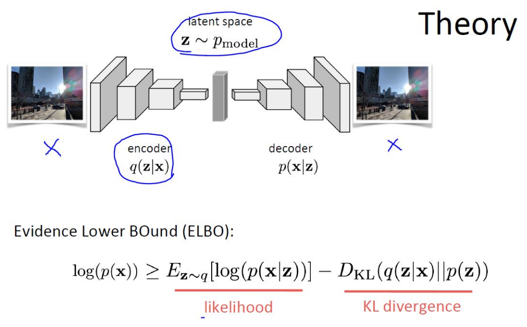

Theory
latent space
$\mathbf{z} \sim p_{\text{model}}$
encoder
$q(\mathbf{z}|\mathbf{x})$
decoder
$p(\mathbf{x}|\mathbf{z})$
Evidence Lower BOund (ELBO):
$\log(p(\mathbf{x})) \geq E_{\mathbf{z}\sim q}[\log(p(\mathbf{x}|\mathbf{z}))] - D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$
likelihood
KL divergence

Figure 17: *ELBO loss*

The equation of the elbow loss can be written as

$$log((p(x))) \leq E_{z\bar{q}}[log(p(x|z)] - D_{KL}(q(z|x)||p(z)) \quad (4)$$

one way to measure the how our model $p(x|z)$ fits the observed data is to measure the KL divergence. KL divergence measures between the two data distribution(normal distribution). This VAE have the encoder part to generate a code and the decoder part that took it back. This is unsupervised because the input image and the output image are almost same so we have ground truth for every image. When we have to generate a new image, we have to draw the code from the latent space and involves the simple to complex distribution. The way the VAE works is to ensure that the distribution of encoder $q(z|x)$ is to be same like latent space distribution **z** to learn the structure of the latent space. So, we need to train in a such a way that these two distribution should match. So, the objective of doing that is miximizing the likelihood of the log(p(x)) as in equ 4 we want to maximize that but is bit difficult so we derive another two term (RHS of the equ 4) First term is the likelihood of decoder $p(x|z)$ and the second term is the KL divergence that is difference between the encoder and the latent space distribution. So, we need to maximize the Right hand side of the equation 4 such that our encoder model correctly encodes the image data and sample it in the latent space Z. BY looking at the equation 4 and figure 18, it can be understood that maximization of the elbow with respect to the parameters

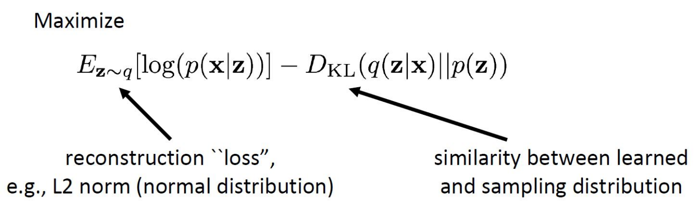

Maximize
$E_{\mathbf{z}\sim q}[\log(p(\mathbf{x}|\mathbf{z}))] - D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$
reconstruction ``loss'', e.g., L2 norm (normal distribution)
similarity between learned and sampling distribution

Figure 18: *ELBO loss*

will optimize the two things we care about.

- It will approximately maximize the marginal likelihood of the $log(p(x|z)$. This means that our generative model will become better.
- It will minimize the KL divergence of the approximation $q(z|x)$ from the true distribution p(z) so $q(z|x)$ will become better.

2) In your own words, explain how the discriminator part of a GAN can be integrated into a VAE.
The implementation of the discriminator in the VAE is explained by the Min max game theory of the Ian Goodfellow. When we look at the maximum likelihood of the equation 4, the reconstructional loss have some L2 losses because of the normal distribution. When we have the L2 loss between the input image and the output image of the decoder, the output comes to be blurry then. So, we introduce a discriminator. This discriminator reads the output of the generator or the decoder part and **try to produce the output between 0 is it's fake image and 1 if it is real one.** If it is blurry, then the discriminator tell it as fake.

- Discriminator D tries to distinguish the real and the fake images.
- The generator G tries to fool the D, tries to make synthetic images look "as real as possible". see the figure 19 for minmax game

$$\underbrace{\min}_{G}\,\underbrace{\max}_{D}\quad \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\log(d(\mathbf{x})) + \mathbb{E}_{\mathbf{z}\sim p_{\text{model}}}\log(1 - d(g(\mathbf{z})))$$

Figure 19: *MinMax game*

The discriminator make the expectance of $log(d(x))$ (first term of the figure 19) as close to one and also when we draw from generator discriminator makes $log(1 - d(g(z)))$ as 1. But the generator will act on the second term and makes it to minimize that by making the $d(g(z))$ close to 1 so that whole second term is close to $log(0)$. This is how the discriminator is implemented in the VAE to get the better loss function.

3) Can you think of a way of using a classifier as part of the loss when training the VAE?

In the advanced part below, we usally train our model network with the MNIST set of images and during test of the classifier with the VAE generated images and labels from the MNIST set, we will calculate the accuracy for the network. This accuracy can we considered as a loss function because we then try to improve the accuracy of the network by adding more training images which helps in setting up the suitable parameters for the digit classification. The basics process of the loss function is to set the suitable parameters for the classififer. So, In this way we can consider the classifer as a loss function when we train our model with the images generated by VAE.

4. ADVANCED PART

In this part of the project, We need to create a training data set for the MNIST dataset, also we need to create digits classification network for our data set and predicting the accuracy for that network, This part is kind of similar to the experiments from lab 2. For this reason , we have created a Better CNN network with two conv and max pooling network. For calculating the accuracy, we compare our test image original labels with the predicted labels from our training network. The accuracy is found to be 0.9847 or around 98%. This is only with the data from the MNIST but further we going to train for our classifier with the images generated from the VAE. So, this we are required to generate a image with the help of decoder from the latent space and we going to train our classifier in a such a way that our training images will have both the generated images and the images from MNIST data set. From the question, we going to investigate for different percentages of the images from MNIST. Initially, we used 20% of our images and showed 96.93% of accuracy. Consecutively, for 50% and 100% images set, the accuracy is 97.97% and 98.69% respectively. Thus, the accuracy is constantly increasing as the data set from the MNIST as increase.

5. Reference

[1] Eric Chu, Variational Autoencoders, https://web.media.mit.edu/ echu/assets/projects/6882/6-882-Final-Report-VAE.pdf

[2] Carl Doersch, Tutorial on Variational Autoencoders. https://arxiv.org/pdf/1606.05908.pdf

[3] Diederik P Kingma, Max Welling, Auto-Encoding Variational Bayes. https://arxiv.org/pdf/1312.6114.pdf

[4] Diederik P. Kingma, Jimmy Lei Ba, ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION https://arxiv.org/pdf/1412.6980.pdf

[5] NingQian, On the momentum term in gradient descent learning algorithms, https://www.sciencedirect.com/science/article/abs/pii/S0893608098001166?via%3Dihub

[6] Nitish Srivastava, Geoffrey Hinton, Dropout: A Simple Way to Prevent Neural Networks from Overfitting https://jmlr.org/papers/v15/srivastava14a.html

[7] Mathworks https://www.mathworks.com/help/deeplearning/ug/train-a-variational-autoencoder-vae-to-generate-images.html