Implementation of Variational Auto-encoder

STAT GR5242 Fall 2018

Yankun Huang, Binhan Wang, Yaoqi Guo, Jiaxi Wu

Dec 16, 2018

Abstract

Variational auto-encoder (VAE) summarizes latent features underneath the given huge dataset. The respective decoder reconstruct the data with small loss. In this project, we first review key points in VAE and then apply it to the SVHN dataset. We also investigate different choices of parameter setting and compare different results. Finally we conclude our work and propose several thoughts for future improvement.

1 Model Overview

1.1 Auto-encoder

In machine learning, we are often given high-dimensional data $\mathbf{X}$ such as images. In some cases, different images share similar features $\mathbf{Z}$ which are viewed as latent variable. Unlike high-dimensional variable $\mathbf{X}$, the latent variable $\mathbf{Z}$ has much smaller dimension. According to Chapter 14 in [1], we can construct an auto-encoder where $\mathbf{Z}$ stands for the hidden layer. We can denote an encoder as q and a decoder as p with $\mathbf{X}' = p(q(\mathbf{X}))$.

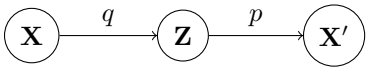

Figure 1: Structure of Auto-encoder

For an input data point $\mathbf{x}$, if the auto-encoder satisfies that the reconstruction loss $L(\mathbf{x}, \mathbf{x}')$ meets our requirement, then we only need to store the auto-encoder and features $\mathbf{z}$ instead of high-dimensional $\mathbf{x}$.

1.2 Evidence Lower Bound

Consider the marginal likelihood of $\mathbf{x}$ controlled by parameter $\boldsymbol{\theta}$

$$p(\mathbf{x}|\boldsymbol{\theta}) = p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) \frac{p(\mathbf{x}|\boldsymbol{\theta})}{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})} = \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})}$$

we have the log-likelihood

$$\ln p(\mathbf{x}|\boldsymbol{\theta}) = \ln p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) - \ln p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$$

Taking expectation with respect to a recognition distribution $q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})$ controlled by parameter $\boldsymbol{\phi}$, we have

$$\begin{aligned} \ln p(\mathbf{x}|\boldsymbol{\theta}) &= \int q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) \ln \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \phi)} \mathrm{d}\mathbf{z} - \int q(\mathbf{z}|\mathbf{x}, \phi) \ln \frac{p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} \mathrm{d}\mathbf{z} \\ &= \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{x}) + \mathrm{KL}(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) || p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})) \end{aligned}$$

Thus we have

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{x}) = \ln p(\mathbf{x}|\boldsymbol{\theta}) - \mathrm{KL}(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) || p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})) \tag{1}$$

Since $\mathrm{KL}(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) || p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})) \geqslant 0$, we get that the evidence lower bound (ELBO) $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{x}) \leqslant \ln p(\mathbf{x}|\boldsymbol{\theta})$ as the left part of equation 2 in [2]. Then our VAE wants to find appropriate $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ such that the ELBO $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathbf{x})$ is

maximized, which is similar to variational inference.
In the equation (1), we have

$$\begin{aligned}\mathrm{KL}(q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})||p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})) &= -\int q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})\ln\frac{p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}\mathrm{d}\mathbf{z} \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}[-\ln p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta}) + \ln q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})]\end{aligned}$$

Note that

$$p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta}) = \frac{p(\mathbf{z},\mathbf{x}|\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} = \frac{p(\mathbf{z},\mathbf{x}|\boldsymbol{\theta})}{p(\mathbf{z}|\boldsymbol{\theta})}\frac{p(\mathbf{z}|\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} = p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})\frac{p(\mathbf{z}|\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})}$$

Thus

$$\begin{aligned}\mathrm{KL}(q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})||p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}[-\ln p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta}) - \ln p(\mathbf{z}|\boldsymbol{\theta}) + \ln p(\mathbf{x}|\boldsymbol{\theta}) + \ln q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})] \\ &= \ln p(\mathbf{x}|\boldsymbol{\theta}) - \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}[\ln p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})] + \mathrm{KL}(q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})||p(\mathbf{z}|\boldsymbol{\theta}))\end{aligned}$$

Substitute it into equation (1) and get the expression for the ELBO

$$\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi},\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}[\ln p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})] - \mathrm{KL}(q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})||p(\mathbf{z}|\boldsymbol{\theta})) \tag{2}$$

which is equation 3 in [2].

1.3 Variational Auto-encoder

By Bayes rule, $p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta}) \propto p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})p(\mathbf{z}|\boldsymbol{\theta})$. However, we cannot simply use $p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$ as the encoder. According to Chapter 14 in [1], if an auto-encoder successfully sets $p(q(\mathbf{x})) = \mathbf{x}$ for every data point $\mathbf{x}$, then it is kind of useless. Hence we can use some $q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})$ which is controlled by other parameters $\boldsymbol{\phi}$ to approximate $p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$. Based on this recognition, we get the VAE as below.

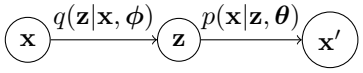

Figure 2: Structure of VAE

2 Model Construction

2.1 SGVB Estimator

Given a data point $\mathbf{x}$ with dimensionality d, the SGVB estimator for ELBO is introduced in section 2.3 of [2]. Here we choose the second version since we can simplify two terms separately. Thus

$$\widetilde{\mathcal{L}}(\boldsymbol{\theta},\boldsymbol{\phi},\mathbf{x}) = \frac{1}{L}\sum_{l=1}^{L}\ln p(\mathbf{x}|\mathbf{z}_l,\boldsymbol{\theta}) - \mathrm{KL}(q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})||p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta}))$$

where the negative cross-entropy $\mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}[\ln p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})]$ is replaced by a Monto Carlo estimate. According to [2], we choose $L = 1$. So the formal SGVB estimator, which consists of two terms, is shown below.

$$\widetilde{\mathcal{L}}(\boldsymbol{\theta},\boldsymbol{\phi},\mathbf{x}) = \ln p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta}) - \mathrm{KL}(q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})||p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})) \tag{3}$$

For the first part, we let $p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})$ be a multivariate Bernoulli distribution as suggested in Appendix C.1 of [2]. Thus

$$\ln p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta}) = \sum_{i=1}^{d}[x_i\ln y_i + (1-x_i)\ln(1-y_i)] \tag{4}$$

where d is the dimension of $\mathbf{x}$ and $\mathbf{y}$ is gotten from the equation 11 in [2].

The other one is $-\mathrm{KL}(q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})||p(\mathbf{z}|\mathbf{x},\boldsymbol{\theta}))$ which is a negative KL divergence. To simplify our model, we select Gaussian distributions $q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi}) = \mathrm{N}(\mathbf{z}|\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))$ and $p(\mathbf{z}|\boldsymbol{\theta}) = \mathrm{N}(\mathbf{z}|\vec{0},\mathbf{I})$ as suggested in Appendix B of [2]. However, the notations in this appendix is confusing and misleading. In this part, we refer to the tutorial [3]. From section 2.2, $\boldsymbol{\mu}(\mathbf{x})$ and $\boldsymbol{\Sigma}(\mathbf{x})$) are trained from the neural network in the encoder. To simplify the computation, $\boldsymbol{\Sigma}(\mathbf{x})$) is constrained to be a diagonal matrix with dimensionality k. Here k is flexible, so we will compare different values of k in the later experiments. Then the KL divergence is

$$\begin{aligned}
\mathrm{KL}(\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))||\mathrm{N}(\vec{0},\mathbf{I})) &= \mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\ln \mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x})) - \ln \mathrm{N}(\vec{0},\mathbf{I})] \\
&= \mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[-\frac{k}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{\Sigma}(\mathbf{x})| - \frac{1}{2}(\mathbf{z}-\boldsymbol{\mu}(\mathbf{x}))^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}(\mathbf{z}-\boldsymbol{\mu}(\mathbf{x})) \\
&\quad +\frac{k}{2}\ln(2\pi) + \frac{1}{2}\ln|\mathbf{I}| + \frac{1}{2}(\mathbf{z}-\vec{0})^{\mathsf{T}}\mathbf{I}^{-1}(\mathbf{z}-\vec{0})] \\
&= \mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[-\frac{1}{2}\mathbf{z}^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\mathbf{z} + \frac{1}{2}\mathbf{z}^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\boldsymbol{\mu}(\mathbf{x}) + \frac{1}{2}\boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\mathbf{z} + \frac{1}{2}\mathbf{z}^{\mathsf{T}}\mathbf{z}] \\
&\quad -\frac{1}{2}\ln|\boldsymbol{\Sigma}(\mathbf{x})| - \frac{1}{2}\boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\boldsymbol{\mu}(\mathbf{x})
\end{aligned}$$

For four parts in the expectation of the above equation, discuss them separately.

$$\begin{aligned}
\mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathbf{z}^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\mathbf{z}] &= \mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathrm{tr}(\mathbf{z}\mathbf{z}^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1})] = \mathrm{tr}(\mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathbf{z}\mathbf{z}^{\mathsf{T}}]\boldsymbol{\Sigma}(\mathbf{x})^{-1}) \\
&= \mathrm{tr}[(\boldsymbol{\Sigma}(\mathbf{x}) + \boldsymbol{\mu}(\mathbf{x})\boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}})\boldsymbol{\Sigma}(\mathbf{x})^{-1}] = \mathrm{tr}(\mathrm{I} + \boldsymbol{\mu}(\mathbf{x})\boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}) \\
&= k + \mathrm{tr}(\boldsymbol{\mu}(\mathbf{x})\boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}) = k + \boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\boldsymbol{\mu}(\mathbf{x})
\end{aligned}$$

$$\mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathbf{z}^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\boldsymbol{\mu}(\mathbf{x})] = \mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathbf{z}]^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\boldsymbol{\mu}(\mathbf{x}) = \boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\boldsymbol{\mu}(\mathbf{x})$$

$$\mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\mathbf{z}] = \boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathbf{z}] = \boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x})^{-1}\boldsymbol{\mu}(\mathbf{x})$$

$$\begin{aligned}
\mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathbf{z}^{\mathsf{T}}\mathbf{z}] &= \mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathrm{tr}(\mathbf{z}\mathbf{z}^{\mathsf{T}})] = \mathrm{tr}(\mathbb{E}_{\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))}[\mathbf{z}\mathbf{z}^{\mathsf{T}}]) \\
&= \mathrm{tr}(\boldsymbol{\Sigma}(\mathbf{x}) + \boldsymbol{\mu}(\mathbf{x})\boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}) = \mathrm{tr}(\boldsymbol{\Sigma}(\mathbf{x})) + \boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\mu}(\mathbf{x})
\end{aligned}$$

Substitute all four terms into the KL divergence and get

$$-\mathrm{KL}(\mathrm{N}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\Sigma}(\mathbf{x}))||\mathrm{N}(\vec{0},\mathbf{I})) = \frac{1}{2}[k + \ln|\boldsymbol{\Sigma}(\mathbf{x})| - \boldsymbol{\mu}(\mathbf{x})^{\mathsf{T}}\boldsymbol{\mu}(\mathbf{x}) - \mathrm{tr}(\boldsymbol{\Sigma}(\mathbf{x}))] \tag{5}$$

Combining equation (4) and (5), we can create the loss optimizer in TensorFlow and apply the stochastic optimization method Adam. According to the introduction in [4], Adam is a gradient-based algorithm which solves minimization problems. Since we want to maximize the ELBO in our VAE, we minimize the negative SGVB Estimator in TensorFlow.

2.2 Neural Networks

As a result, we build the neural network of the VAE according to [3]. Nodes for sampling $\boldsymbol{\varepsilon}$, Hadamard product $\circ$ and $+$ stand for the reparameterization trick in section 2.4 in [2].

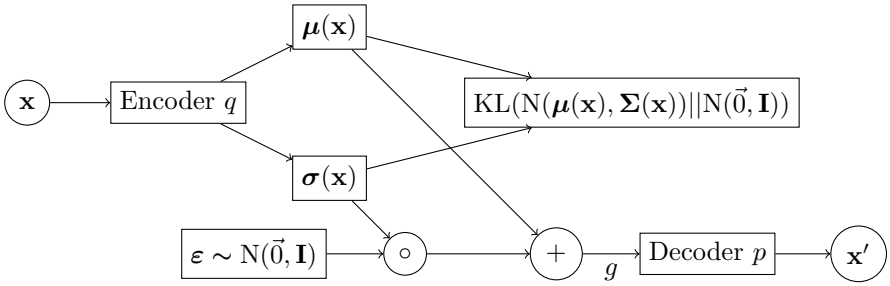

Figure 3: Network of VAE

Here $g(\boldsymbol{\varepsilon}, \mathbf{x}) = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\sigma}(\mathbf{x}) \circ \boldsymbol{\varepsilon}$. In the encoder $q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})$, we adopt the structure suggested in Appendix C.2 in [2] and modified it by adding one more layer. Then the neural network is as follow.

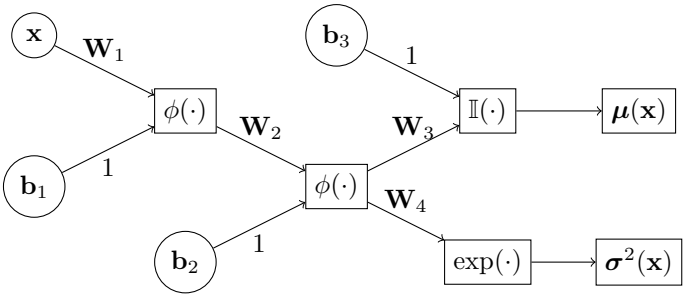

Figure 4: Network of Encoder q

Note that in this network, we get $\boldsymbol{\sigma}^2(\mathbf{x})$, the vector of variance for each dimension of $\mathbf{z}$. Since $\boldsymbol{\Sigma}(\mathbf{x})$ is constrained to be a diagonal matrix with dimensionality k, we get $\boldsymbol{\Sigma}(\mathbf{x}) = \mathrm{diag}(\boldsymbol{\sigma}^2(\mathbf{x}))$. Also, we modified the structure given in [2] by deleting a bias node before $\exp(\cdot)$. Also, we add one more layer and change the activation function to $\phi(\cdot)$. In the decoder $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$, we adopt the structure suggested in Appendix C.1 in [2] and modified it as same as what we do in the encoder. Then the neural network is as follow.

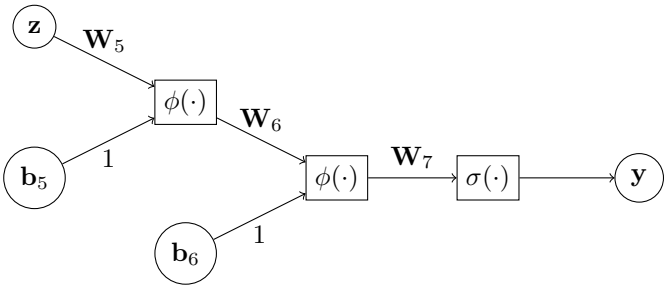

Figure 5: Network of Decoder p

Thus $\mathbf{x}' \sim$ Multivariate Bernoulli($\mathbf{y}$).

3 Experiment and Outcome

3.1 Tools

In this section, we discuss the details about our implementation in TensorFlow and our model performance. We modified an existing VAE model located at https://jmetzen.github.io/2015-11-27/vae.html. This model was implemented on the MNIST dataset which contains purely black and white, handwritten digits. We improved the weight initialization process to avoid unnecessary computation. We also created additional train procedure to apply this algorithm on RGB level.

Since we are implementing this model on SVHN dataset, we want our dataset to have the same attributes as

the MNIST. We get insight from previous implementation, and construct a class object of our dataset. The original code reference can be found here: https://github.com/wbh0912/AML-Final-Project.

3.2 Dataset

The SVHN dataset is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It has 73257 digits for training and 26032 digits for testing. Each data point is a 32×32 png image, so the dimension of each input is $d = 32 \times 32 = 1024$. It can be viewed similarly in flavor to the MNIST (e.g., the images are of small cropped digits). It is obtained from house numbers in Google Street View images.

In this project, for convenience we only use the cropped images whose matrices have the same dimensions. In Figure 6 given below, we can find that there can be multi-digits in one image. The cropping does not successfully separate all individual digits. As a result, different digits in the same image can mutually interact. This may lead our model to learn false information. Besides, some of the digit images are fuzzy and not clearly readable. As a result, we do not expect all decoded digits to be clearly enough for recognition. Here we list some samples as following:

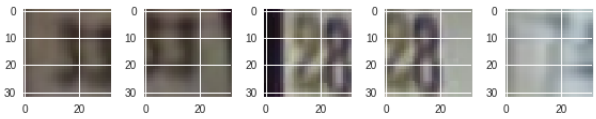

Figure 6: Data Preview

Starting from the left side, we can find that digits in the first two images are fuzzy and those in the third and fourth images are relatively clear. However, the last image hardly shows any information. Also the third and fourth images actually represent the same digits.

3.3 Implementation and Outcomes

Since the referred VAE model was built to deal with black and white images in the MNIST, it takes 2-D arrays as input. To apply VAE model to our color images in the SVHN on RGB level, we divide each image into three channels and train models separately. Finally, we combine three reconstructed arrays together to get decoded images.

In model construction, we claimed that the choice of dimension of latent space k is flexible. So firstly we compare the results under $\mathbf{z}$ with different dimensions and list some outcomes as following:

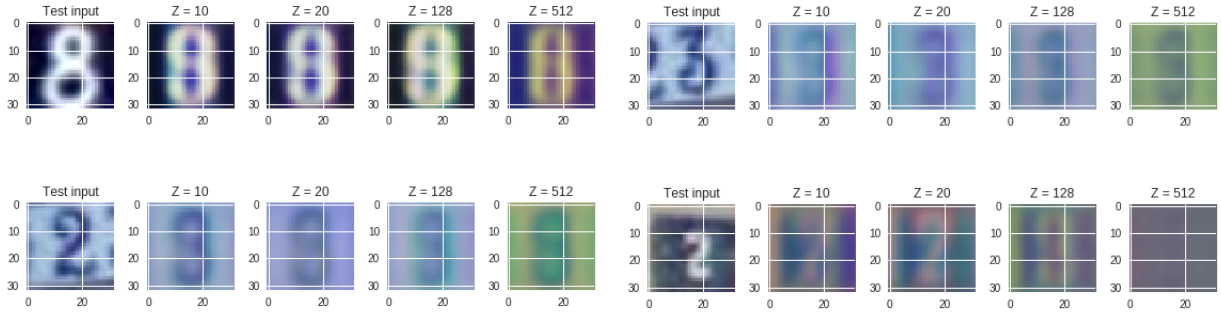

Figure 7: Outcomes with Different k

From this comparison, we can see that increasing of k does not lead to more accuracy on the decoded images. In fact, as we increase the dimension of $\mathbf{z}$, the image became more and more blurry. Some of the images which are already readable, e.g. the first sample in Figure 7 that represents 8, is transformed to something like 9 or 6 with this increasing. We interpret that the digits in each image do not have that many latent features. Hence increasing the dimension of $\mathbf{z}$ will lead to confusion and worse performance.

Next we apply different dimensions of the input with respect to $\phi(\cdot)$ nodes in Figure 4. Since the data point $\mathbf{x}$

is a $d \times 1$ vector and the outputs are $k \times 1$ vectors, we have two free parameters according to the dimensions of weight matrices. In the encoder, we set $\mathbf{W}_1$ as a $m \times d$ matrix and $\mathbf{W}_2$ as a $n \times m$ matrix. Thus dimensions of $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{W}_3$ and $\mathbf{W}_4$ are $m \times 1$, $n \times 1$, $n \times k$ and $n \times k$ respectively. Symmetrically there are also two free dimension parameters in the decoder. For the convenience of construction, in Figure 5 we set $\mathbf{W}_5$ as a $n \times k$ matrix and $\mathbf{W}_6$ as a $m \times n$ matrix. Thus dimensions of $\mathbf{b}_5$, $\mathbf{b}_7$ and $\mathbf{W}_7$ are $n \times 1$, $m \times 1$, $d \times m$ respectively.

In this part, we set $m = n$. Then we compare the results under different choices of m and list some outcomes as following:

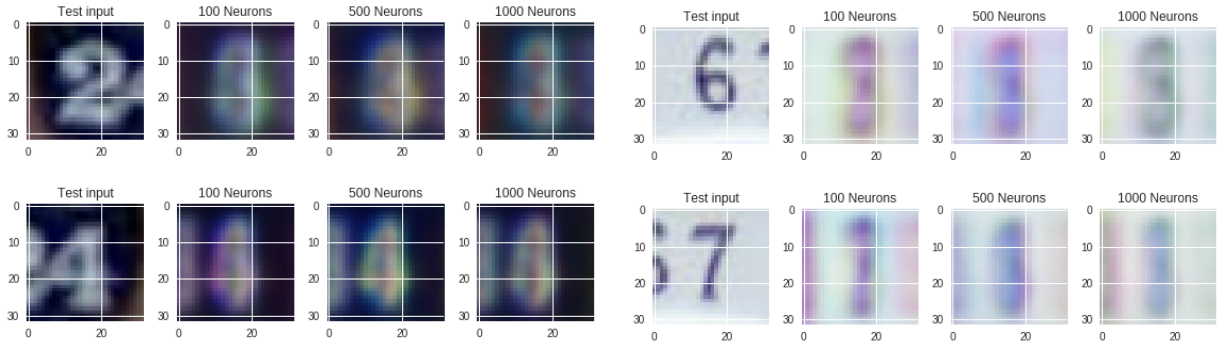

Figure 8: Outcomes with Different Choices of m

From this comparison, we can see that neither decreasing nor increasing m can significantly enhance our model performance. From the listed outcomes, 500 seems to be a reasonable choice for our current VAE model.

Then we set m and n differently and implement the model. We use the notation $m - n$ in the following samples.

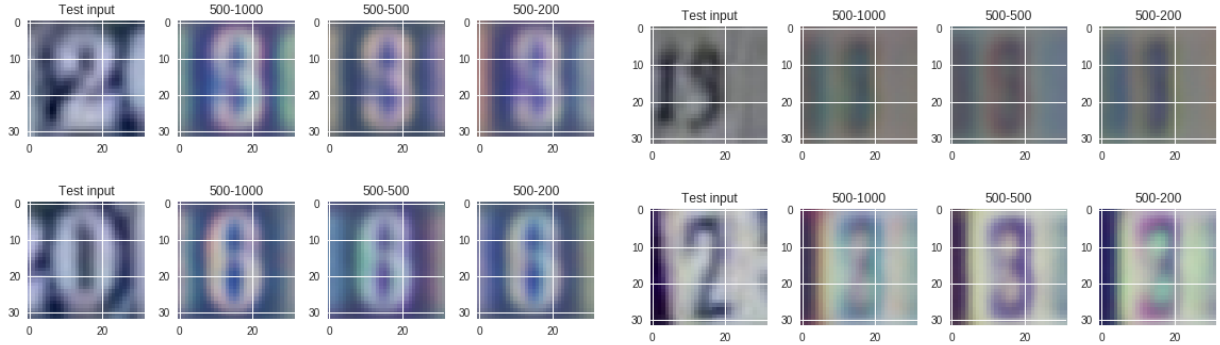

Figure 9: Outcomes with Different Choices of m and n

Things seem to have little change. The samples show that our model misclassifies many digits.

At last we compare different settings of epoch. From three selected results in Figure 10, we can find that the whole cost converges when the epoch increases. However when epoch = 50, the running time is very long.

```
Epoch: 0010 cost= 643.852019701
Epoch: 0020 cost= 642.582284242
Epoch: 0050 cost= 641.465180082
```

Figure 10: Outcomes with Different Epoches

Then the corresponding outcomes are as following:

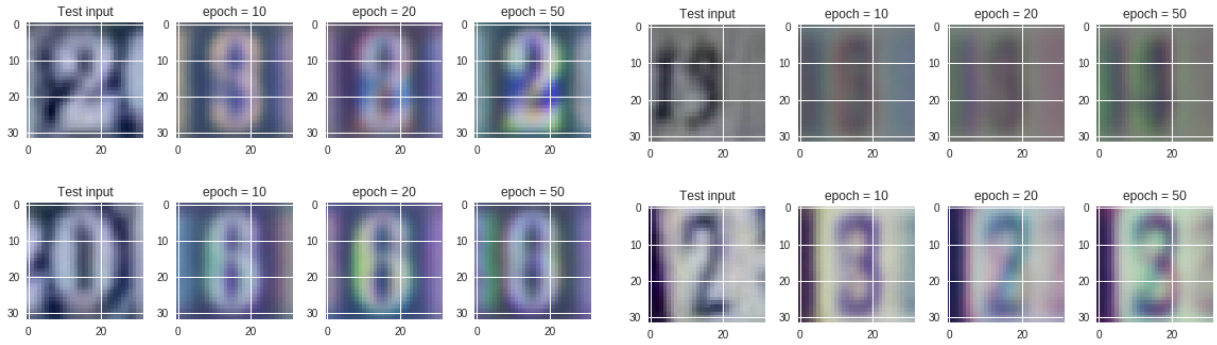

Figure 11: Outcomes with Different Epoches

For batch training, all of the training samples pass through the model simultaneously in one epoch before weights are updated. Here we can see when we increase the number of epoch in the training process, some of the outcomes becomes clearer and more accurate. In these samples, digits in the first and third ones are misclassified in the training process, but finally they are rightly recognized. However, there are many counterexamples. The digit in the last sample shows that our model is confused when recognizing 2 and 3 during the training process.

4 Conclusion and Future Work

4.1 Summary and Conclusion

Variational auto-encoder is a promised generative model. From our results, we can see that we can generate a fair amount of desired images. For those inaccurate outcomes, they still roughly follow the same patterns as the original images. We also obtain some valuable information while exploring the effects of different settings.

1. Dimension of latent feature $\mathbf{z}$. Since in section 1.1 we state that VAE model presents dimension reduction, a high-dimensional latent space does not make sense intuitively. According to our implementation, we conclude that 10 or 20 is a reasonable choice for the dimension of latent space k with a input of length $d = 1024$ (32 by 32 matrix). The optimal k may subject to the input dimension d.

2. Choices of width m and n for hidden layers. According to the outcomes, setting $m = n$ or $m \neq n$ does not lead to much difference. Also changing m or n respectively does not significantly enhance the performance. The optimal m and n are vague and we can hardly get conclusion from our experiments.

3. Number of epoch. Clearly increasing training time can enhance the VAE model. From the sample outputs, we find that digits generated from model with epoch = 50 seem more similar to the original ones. Also we should note that more epochs will cause more computation and increase the training time. If time allows, we should try a larger number of epoch.

Due to several reasons, the model we have so far has not reached all of our expectations.

1. The SVHN data experience cropping and thus contain less information. As we can see from the samples in section 3.2, some of the digit images are blurry. In addition, encoding and decoding procedure definitely cause some loss of information. Thus it is reasonable that the outcomes are less readable.

2. This model is kind of naïve. While implemented on the MNIST dataset, the model produces decoded images which are still clear enough to read. Since the training images are black and white, the decoded images still contain much information. When we apply the same model to the SVHN dataset on RGB level, we can assume that the information loss is at least tripled and thus we obtain more blurry outcomes.

3. We trained the model for three channels individually, which might not be the right way. Maybe we should treat an image as an ensemble regarding the training and optimizing procedures.

4.2 Future Work

Then we list several points for the future work according to our discussion in the previous section.

1. We plan to look for datasets which contain images with higher resolution and better cropping.

2. We plan to modify the network architecture and try other activation functions such as $\tanh(\cdot)$ and $\sigma(\cdot)$ so that the VAE can more efficiently encode a RGB image input.

3. We plan to add labels as a conditional variable to construct a conditional VAE model. In this way, we can enhance its accuracy and generate images with specification. We hope that this step can improve our model performance.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Cambridge: MIT press, 2016.

[2] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[3] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[4] Diederik P. Kingma and Jimmy Lei Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv: 1412.6980*, 2014.