2AMU20 Generative Models
Homework Assignment 2

Yang Yang, 1552139 Iason Theodorou, 1618377

Group 9

Task 1: Implement and Train a VAE

In this assignment we will implement a variational auto encoder (VAE) using (de)convolutional neural networks ((d)CNN) using *pytorch* library, so to perform several image analysis tasks to the MNIST dataset of digits. We consider 50.000 training image dataset, 10.000 validation dataset and finally the last 10.000 image dataset for test purposes. The images have 28×28 dimensions and the pixel values are normalized in the range [0,1]. We transform the dataset into tensors and we pass them through a torch data-loader so to create data samples with batch-size of 256.

1a: Model Setup

VAE is a latent variable model. It describes higher dimensional data into smaller dimensional latent variable space, these latent variables are denoted as $Z = (z_1, z_2, ..., z_n)^T$.

VAE consists of 2 parts: encoder and decoder. Both of these parts are neural networks with similar architectures. The encoder learns the distribution the parameters $\mu = (\mu_1, \ldots, \mu_n)^T = \mu(\mathbf{x})$ and $\sigma = (\sigma_1, \ldots, \sigma_n)^T = \sigma(\mathbf{x})$ in order to learn the probability distribution $q_\phi(Z \mid X)$ where $\mu(x)$ is the mean parameter and $\sigma(x)$ is the standard deviation of the given distribution, which can be expressed as $q_\phi(Z \mid X) = \mathcal{N}\left(\mathbf{z} \mid \mu(X), \left(\sigma^2(X)\right)\right) = \prod_{i=1}^{D} N\left(z_i \mid \mu_i, \sigma_i^2\right)$. In our model the encoder consists of 3 convolutional (conv) layers. The first conv layer has dimensions $(1, 32)$ (1 is the number of channels of input) with a kernel size of (5×5), stride $(1, 1)$ and padding of $(2, 2)$. We pass the conv layer through a rectified linear unit (relu) non-linear function and a pooling layer (pool) of size 2. The second conv layer is of size $(32, 64)$ with exactly the same dimentions of kernel,stride and padding. Again we use a relu and a pooling layer. The last layer is of size $(64, 128)$. We reduce the kernel size on $(3, 3)$ and padding on $(1, 1)$ since now we consider deeper representations of the pictures. Finally two linear layers are used of $(128 * 3 * 3, \textit{latent space})$ dimensions. Those layers are used for learning the μ and the log-variance. Note that it is impossible to train a model using backpropagation, since some parts of it depend on random sampling, and every step of the neural network needs to be differentiable. Therefore, to obtain the latent variables Z, we need to use reparametrization. Essentially, we sample from a variable ϵ, where ϵ is white Gaussian noise $N(\mathbf{0}, \boldsymbol{I})$ and we create the standard deviation from our learned log-variance. We multiply element-wise the ϵ with $\sigma(X)$ and add the $\mu(X)$ to it to make all the steps of our model differentiable. By reparametrize our z variable we now can learn the probability distribution of our latent space $q_\phi(Z \mid X)$.

Now that we obtain our latent variables we can the decoder so to able to generate data. The decoder is used to learn the parameters $(m_1, \ldots, m_n)^T = \mathbf{m}(\mathbf{z})$ and $(s_1, \ldots, s_n)^T = \mathbf{s}(\mathbf{z})$ of the distribution $p_\theta(X \mid Z)$ in order to learn the probability distribution $p_\theta(X \mid Z)$, which can be expressed as $p_\theta(X \mid Z) = \mathcal{N}\left(\mathbf{x} \mid m(Z), \left(s(Z)^2\right)\right) = \prod_{i=1}^{D} N\left(x_i \mid m_i, s_i^2\right)$. Considering our model, as we mentioned earlier our latent variables flow from the encoder through the decoder, which has the inverse architecture from what our encoder had. *Pytorch* has a method that essentially creates transpose convolutional layers which starts with an input of the dimensions of our latent space and outputs 128 dimensions. Also we use kernel size of (5×5), stride of $(2, 2)$ and padding

of $(1, 1)$. Mote that except the conv layer dimensions we keep the other parameters same in every de-conv layer. Also for every output of the de-conv layer we also use a batch normalization and a relu layers before we pass the output to the next layer. The second layer has dimensions of $(128, 64)$, the third has dimensions of $(64, 32)$ and the last one returns the the number of channels of the images. Finally we flatten our tensor so to feed them to two linear layers, one for learning the m and one for learning the $log-variance$. The dimensions of those layers are $(31 * 31, 28 * 28)$. Now that we obtain our sample space, we implement a reconstruction method so to be able to sample from our learnt distribution. This method creates the standard deviation σ from the learned log-variance and we add random noise to it. Every sample is created from a sigmoid function so to be mapped in the range-space of [0,1], with the parameter of $(\mu + \sigma * epsilon(noize))$.

For the training, the loss function is expressed as an optimization (maximization) problem of the conditional log-likelihood, or the lower bound of it (ELBO) since it is very costly to compute the log-likelihood. More will be discussed in the section . We also use an Adam optimizer 0.001, since it is a robust optimizer commonly used for this kind of task. As we mentioned before we used a batch size of 256, latent space dimension of 16.

1b: Model Training

Before discuss the training procedure let us discuss more about the ELBO loss function that is used in our model. As we refereed in the previous section the log-likelihood for a VAE in given by this formula:

$$L = \sum_n log \int p(z^D)p(x^D|z^D; \Theta)\, dz^D$$

Since the integrals are very expensive to compute which is a lower bound of the marginal log-likelihood. ELBO can be derived from this formula:

$$L \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x} \mid \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x})}\right]$$

which is a lower bound approximation for the true log-likelihood. It is proven that for maximizing the ELBO, meaning to be as close as possible to the true value, we need to minimize the KL-divergence since the difference of the $L - ELBO = KL(q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{Z}|\mathbf{x}))$. From that we use the equation bellow:

$$\log p(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{z})] + KL(q(\mathbf{Z} \mid \mathbf{x}) \| p(\mathbf{Z}|\mathbf{x}))$$

Important to notice is that maximizing the ELBO is equivalent as minimizing the negative ELBO and from a technical point of view we use this approach.

In the above section we specified the architecture of our model and the parameters used and in this section we discussed about our loss function. Now we are ready to train the model. We use are dataloader to generate batches and we train our model with our training dataset. We also use the validation dataset after our training is finished for each epoch as shown in Figure 1. The results for 20 epochs we can see that is not ideal. We can see that the negative ELBO seems to be reduced overtime but it is not stable, resulting to many "spikes" in the related plot.

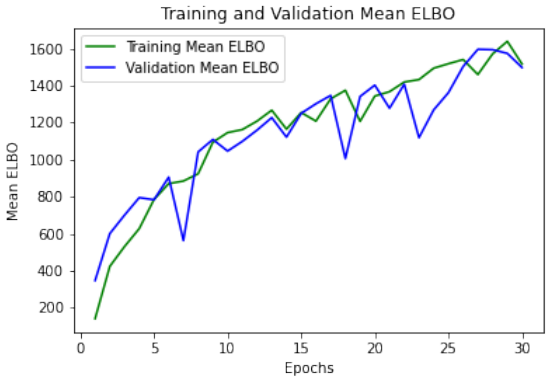

Figure 1: ELBO for training and validation dataset

1c: Stopping Criterion

In this section we implemented a stopping criterion, so to avoid for our model to be trained without improving. Stopping criterion is commonly used in many machine learning applications according to the existing literature and it is very useful for avoid over-fitting or spending resources for a model that is not further improving. We create a function that stores the best negative ELBO result and if this result not changes for some value that we choose to model stops the training. In our case we set the patience variable to 5.

1d: Model Testing

For the last part of task 1 we use our trained model to the test dataset so to evaluate its performance. We use our reconstruction function so to extract the sampled images. We plot the first 32 images from the testing data, side-by-side with their respective reconstructions as shown in Figure 3

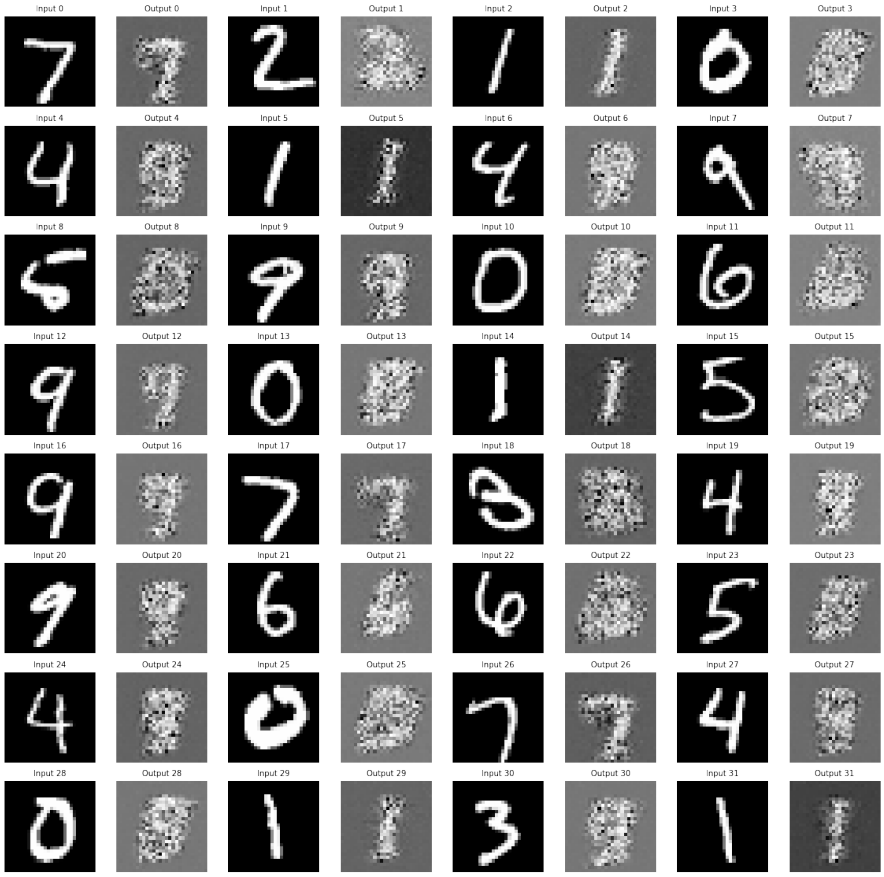

Figure 2: Reconstructed images from the test dataset

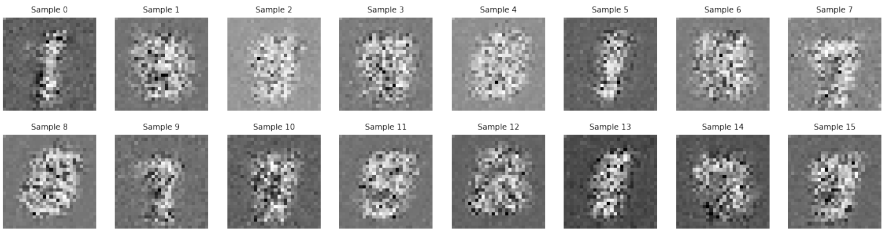

Figure 3: Samples from the Normal distribution

We can see that our model manages to reconstruct some images. On the "easy" digits like $0, 1, 7 or\ 9$ it manages to reconstruct well but with some noise. In cases of more complicated digits like 4 it mostly confuses

them with some other digit. In some difficult case our model responds well for example reconstructing the digit "2". Overall we can see that the model performs average and there is room for improvement.

Task 2: Choice of Conditional Data Distributions

Task 2a: Reflection

In the previous task we implemented a VAE for our given purpose and we use Gaussian distribution for the $p(\mathbf{X}|\mathbf{z})$. From a practical point of view we saw that the results were not sufficient for using this algorithm as a stable version for reconstructing images. Therefore the motivation for trying different distributions for our model seems a reasonable choice. Note that a good aspect of Gaussian distribution is that it is easy to implement because of the reparametrization trick that we used in the previous section, so it was reasonable to start our implementation with this distribution. From a more theoretical point of view also we can see that since we are working with grey-scale images (close to just black and white), which have vector values in the range of [0,1], using distributions that sample from this range-space (e.g Beta or Bernoulli Distribution) would possibly introduce better results.

Task 2b: Beta Distributions

In this task we are going to manipulate our model so the sample space to be a Beta distribution and $p(\mathbf{X}|z)$ to be an (independent) product of Beta distributions. Beta distribution is a continuous probability distribution defined on the interval [0, 1], parameterized by two positive shape parameters, denoted by and β that appear as exponents of the random variable and control the shape of the distribution. In our setting we do not have to modify our model since Beta Distribution requires learning two parameters and we already have two linear layers on our decoder. We change the distribution of our sampling method into Beta since we used to sample from a Gaussian Distribution. Finally the loss function should be reconstructed since our objective now is to maximize the log-likelihood of a Beta distribution. We again maximize ELBO so to approximate the log-likelihood but we now consider the probability density function (PDF) of a Beta distribution, which is given by this formula:

$$PDF = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)},$$
$$where,\ B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta}{\Gamma(\alpha+\beta)}$$

So since the likelihood is a product of Beta distributions that log-likelihood can be calculated as the sum of log Beta distributions and can be obtained by the formula bellow:

$$logp(\mathbf{x}) = \sum(\alpha-1)x(\beta-1)x - \Gamma(\alpha) - \Gamma(\beta) + \Gamma(\alpha+\beta)$$

We set the latent dimension to 16 and we use Adam optimizer with learning rate 0.001 to train the VAE model. We also apply the early stopping mechanism with *patience=5*. The results of the ELBO can be shown in Figure 4. Furthermore we can see some image reconstructions in Figure 5 We can see that ELBO introduces much better results compared to the model we use in task 1. Also the reconstructed images are now much better and we can see that our model can create identifiable digits for the given data points.

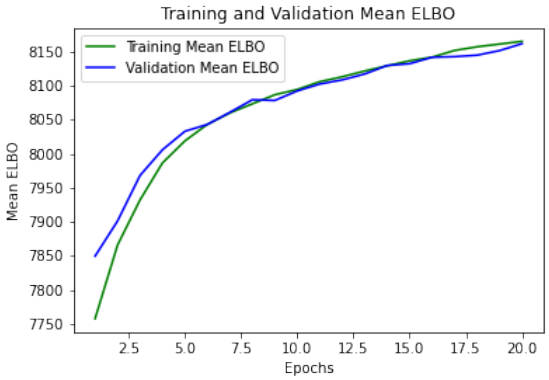

Figure 4: ELBO for Beta Distribution on training and validation dataset

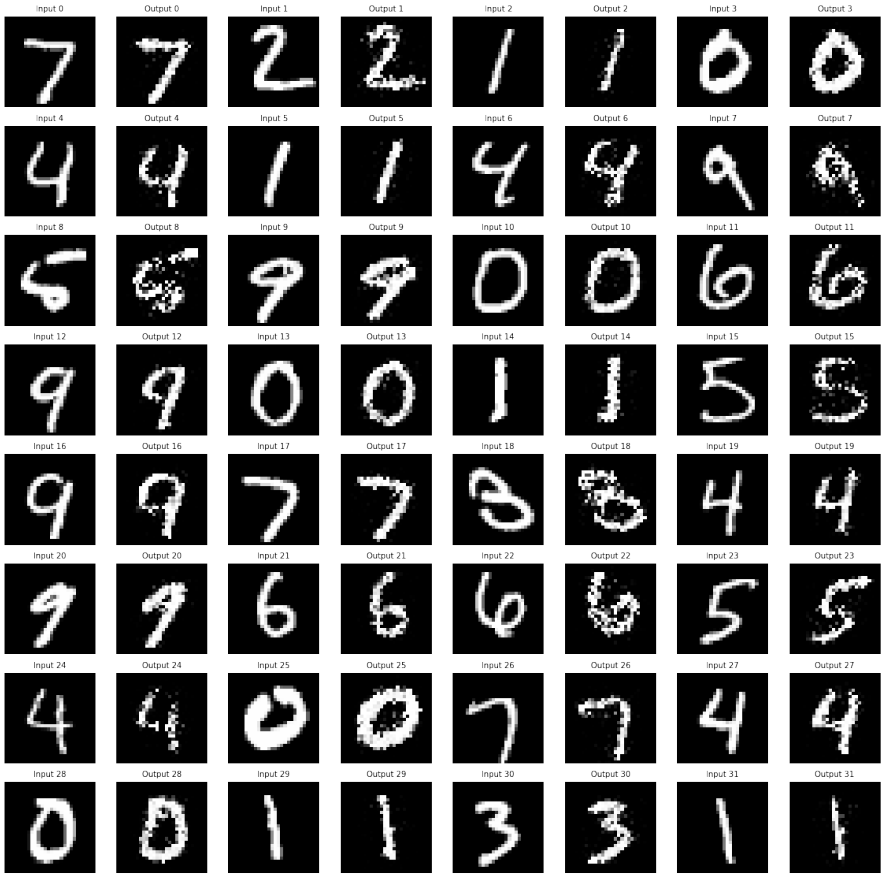

Figure 5: Reconstructed images from the test dataset for Beta Distribution

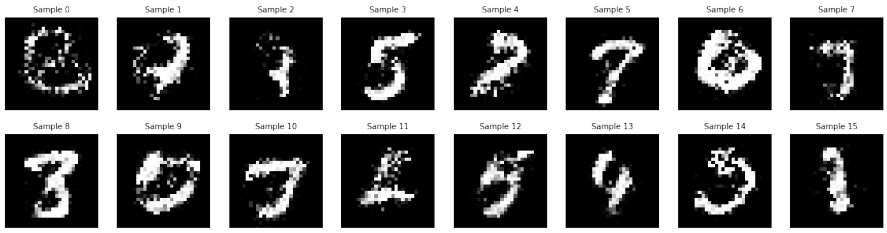

Figure 6: Sample images from Beta Distribution

Task 2c: Categorical Distributions

In this task, we will use the Categorical distribution to construct $p(\mathbf{X} \mid \mathbf{z})$. We discretized the pixel values of the input image into $k > 1$ bins. Let $\mathbf{X}$ denote the random variables over these discrete data points, so that $p(\mathbf{X} \mid \mathbf{z})$ is an (independent) product of Categorical distributions over k outcomes, i.e. such that

$$p(\mathbf{X} \mid \mathbf{z}) = \prod_{d=1}^{D} p\left(X_d \mid \mathbf{z}\right) = \prod_{d=1}^{D} \operatorname{Cat}\left(X_d \mid \boldsymbol{\pi}_d(\mathbf{z})\right) \quad \text{for all } \mathbf{z} \in \mathcal{Z}$$

where $\boldsymbol{\pi}_d(\mathbf{z})$ is a vector containing the parameters of the Categorical distribution over k outcomes, for the d -th feature of the data, as generated by the decoder network when evaluated in $\mathbf{z}$. Then we calculate (conditional) log-likelihood by

$$\begin{aligned} \log p(\mathbf{X} \mid \mathbf{z}) &= \log \prod_{d=1}^{D} p\left(X_d \mid \mathbf{z}\right) \\ &= \sum_{d=1}^{D} \log \operatorname{Cat}\left(X_d \mid \boldsymbol{\pi}_d(\mathbf{z})\right) \quad \text{for all } \mathbf{z} \in \mathcal{Z} \end{aligned}$$

By plugging the probability mass function of Categorical distribution, we have

$$\log p(\mathbf{X} \mid \mathbf{z}) = \sum_{d=1}^{D} \log([x_d = 1] \cdot p_1 + \cdots + [x_d = k] \cdot p_k)$$

where $[x = i]$ is the Iverson bracket and p_i is the event probability that $x = i$.

In implementation, we set $k = 4$ and latent dimension to 16. We use the *Adam* optimizer with learning rate 0.001 to train the VAE model. We also apply the early stopping mechanism with *patience=5*. The graph of the ELBO is shown in Figure 7. Some reconstructions and samples are shown in Figure 8 and 9

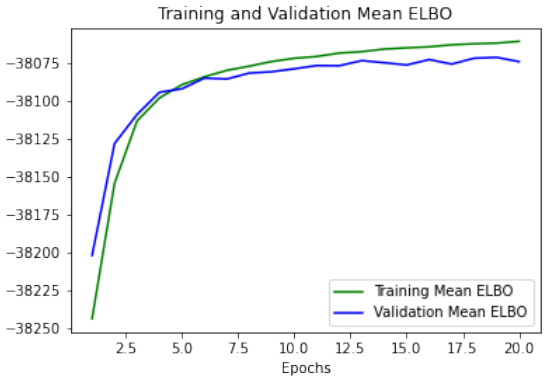

Figure 7: ELBO for Categorical Distribution on training and validation dataset

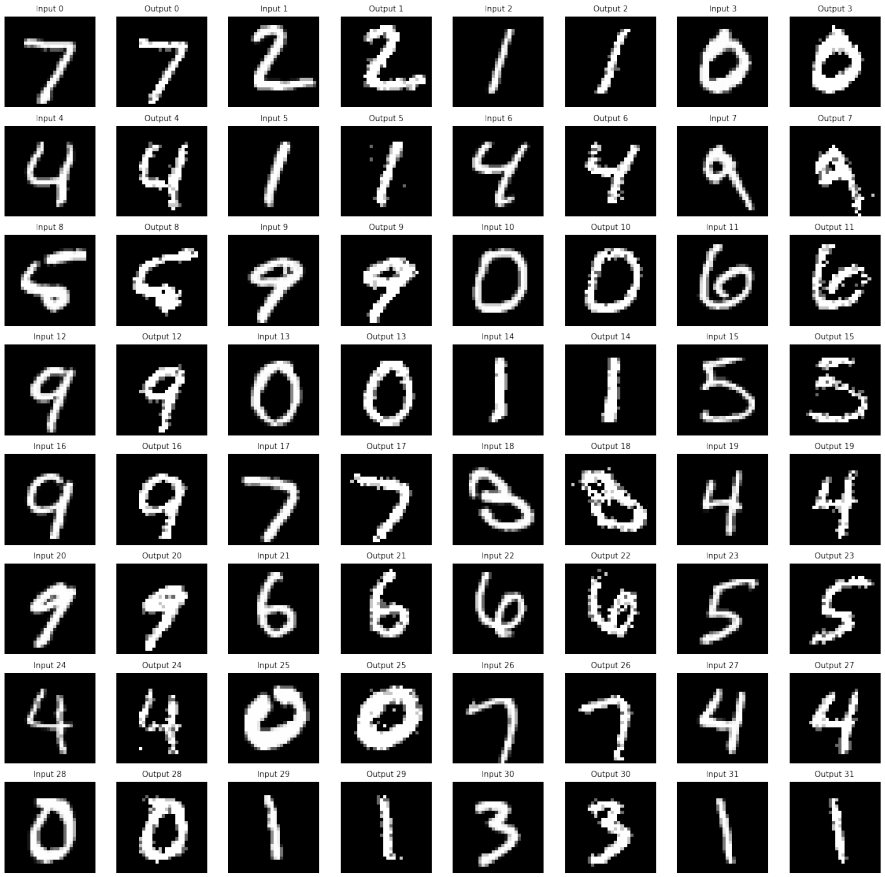

Figure 8: Reconstructed images from the test dataset for Categorical Distribution

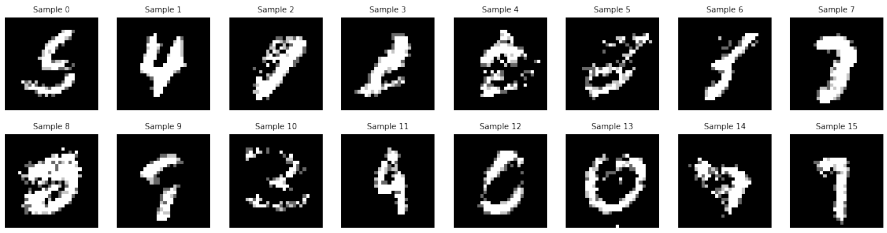

Figure 9: Sample images from Categorical Distribution

Task 2d: Bernoulli Distributions

Finally we will repeat the same procedure with the previous subsections by changing the model to a distribution $p(\mathbf{B}|z)$ over the space $\{0,1\}^D$ of binary images, as an independent product of Bernoulli distributions. Bernoulli distribution is the discrete probability distribution of a random variable that takes the value 1 with probability p and the value 0 with probability $q = 1 - p$. We firstly modify the sample function that we use for reconstructing the images, so to sample from a Bernoulli distribution. Note that we do not use our sample function since we get better results by instead using the posterior means (the output of the decoder), so to reconstruct the images. Also now the decoder contains only one linear layer (not two) since we only need to learn one parameter for Bernoulli distributions. Also we need to consider that the loss function is going to change since the log-likelihood for different distributions is calculated from different formulas. For maximizing the log-likelihood essentially we need to maximize the log of Bernoulli distribution or minimizing the negative log of Bernoulli distribution. Binary Cross Entropy function can maximize the Bernoulli distribution and it is given from the equation below:

$$H(p(\mathbf{B}|\mathbf{z}), p(\mathbf{B}|\mathbf{x})) = \sum_{d=1}^{D} x_d log(\pi_d(\mathbf{z})) + (1 - x_d) log(1 - \pi_d(\mathbf{z}))$$

So we maximize ELBO by minimizing the negative Binary Cross Entropy. The KL-divergence formula of ELBO remains the same. We train our model and we display the ELBO in Figure 10. We can see that this approach introduces the best results until now with both training and validation ELBO to increase drastically. More evaluation of our results can be shown in Figure 11. We can see that now we can barely identify which results are reconstructed and which results are from our data. Our model constantly recreates the images almost perfectly.

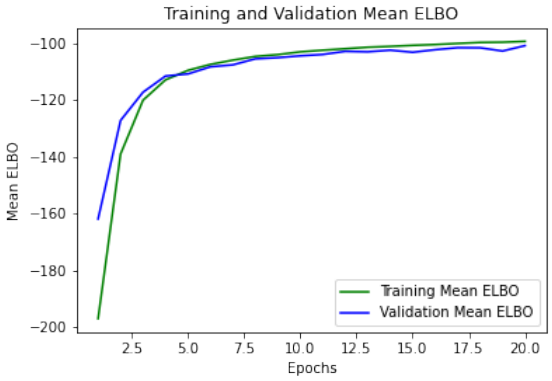

Figure 10: ELBO for Bernoulli Distribution on training and validation dataset

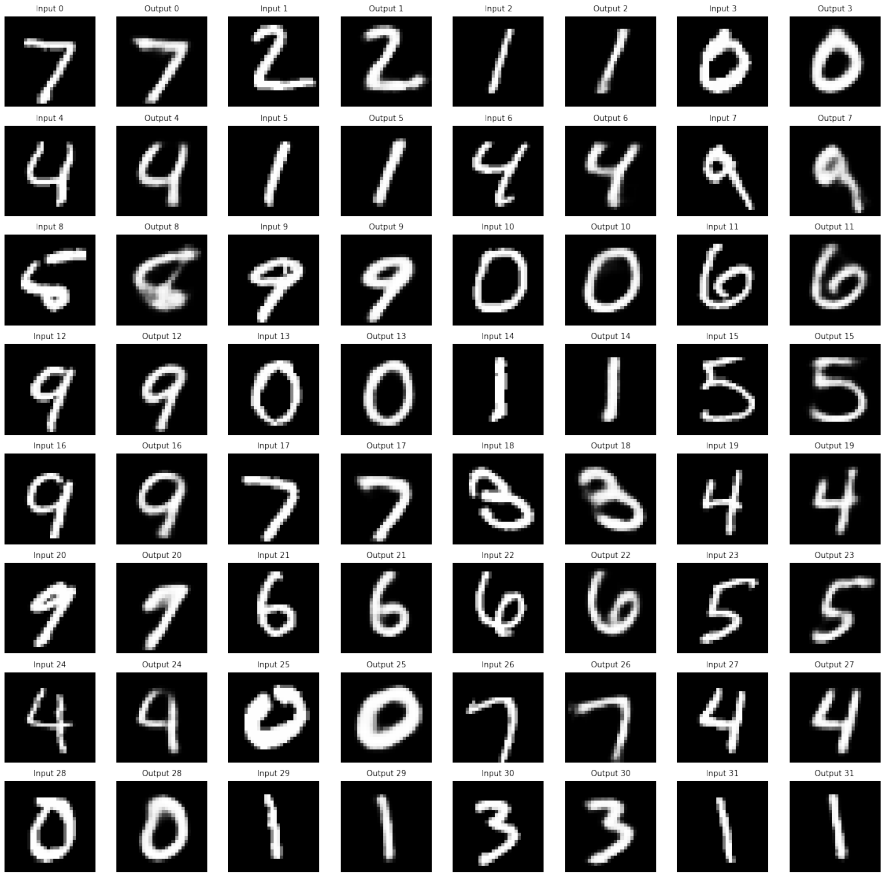

Figure 11: Reconstructed images from the test dataset for Bernoulli Distribution

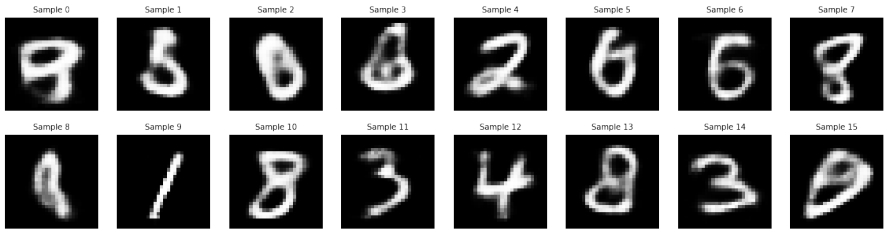

Figure 12: Sample images from Bernoulli Distribution

Task 2e: Further Reflection

Finally we are going to discuss and evaluate the variations of distributions that we used for our model. We tested Gaussian distribution, Beta distributions with [0, 1], Categorical distributions with discretised (binned) data,and Bernoulli distributions with re-interpreted data as probabilities for a given pixel to be black or white. In this specific task Bernoulli distribution (BD) is the most sufficient choice. Bernoulli distribution makes sense for black and white (i.e. binary) images. The Bernoulli distribution is binary, so it assumes that observations may only have two possible outcomes and this matches our dataset specification. Generalizing this statement we can see that when we use distributions with range [0, 1] we obtain better results because of the "nature" of our data. Other distributions that could possibly be effective for our model would be distributions that are in the same space as our pixel values.

Task 3: Investigate the Latent Space

For this task, we will investigate how our model perceives the structure of the latent variables Z, and see how it captures structure that was implicitly present in the data. We will use the Bernoulli-distribution model from Section , since it introduced the best results.

Task 3a: Visualizing the Latent Space

In the first section we will create a two-dimensional latent space and we will train our model as we used on the previous parts. Then we evaluate the model in the first 1000 datapoints of our testing data, by using the encoder. We create a function that plots the $\mu(z_i)$ outputs of our encoder on a 2-dimensional plot, color-coded based on the labels of the digits y_i as shown in Figure 13

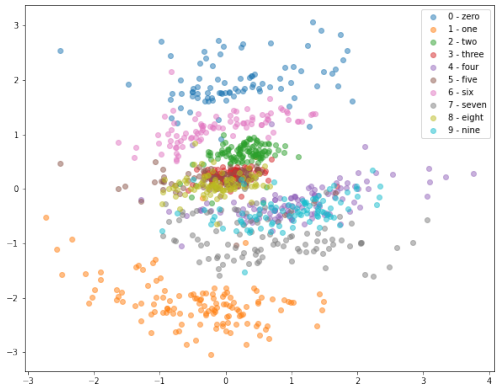

Figure 13: Plotting μ on 2-dimensional latent space

We can see that the model mostly learns to address the difference between the digits. Two cases are somehow interesting to talk about, which are the cases that two classes that overlap which is the figures $4, 9$ and $3, 8$. Those pairs make intuitive since since their figures are very similar, so someone can expect that in lower dimensions their separation is not an easy task. In higher dimensions, these problems should reduce.

Task 3b: Visualizing the Latent Space, using PCA

Now we will train our model on the latent space which is K-dimensional and we will reproduce the same procedure as before. In our setting we specify the dimensions to be 16. When the model is trained we use 1000 test data points and we use the encoder so to store its $\mu(z_i)$ output. Then we perform dimensionality reduction by using principality component analysis (PCA). PCA is a standard technique which is based

on singular value decomposition (SVD) and it can utilized by many different libraries written for python programming language. We use the function of sklearn's library and we we suppress the dataset to its 2 most important components. We plot again the results with respect to the labels as we can see on the Figure 14 bellow:

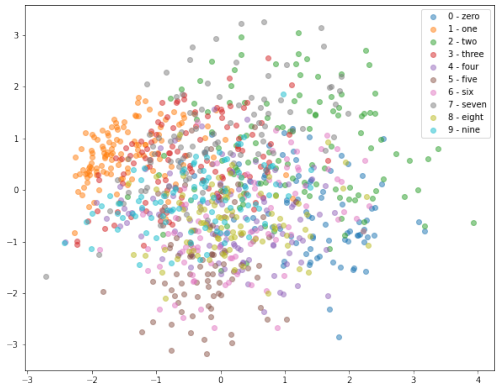

Figure 14: Plotting μ on 16-dimensional latent space using PCA with 2 components

We can see that now the results are not clearly separate and mostly we cannot distinguish between classes in the 2-dimensional space. This does not mean that our model performs purely rather that PCA cannot reduce the dimensions of the latent space and preserve the space separation of the data points. This is quite reasonable since PCA should be used mainly for variables which are strongly correlated. If the relationship is weak between variables, PCA does not work well to reduce data and since our covariance matrix is diagonal we can see that there exist no correlation between our variables.

Task 3c: Interpolation in the Latent Space

For the final part of task 3 we will use linear interpolation so to obtain a sample $\mathbf{Z}$ from our decoder rather than directly reconstruct images, as we did in the previous chapters. Linear interpolation is a method of curve fitting using linear polynomials to construct new data points within the range λ of a discrete set of known data points. So by predicting the values between the different points of our latent sample we aim to produce a reconstructed image. We create a function which takes as inputs to different data points of different class label and the λ value, which represents the range of discrete points that our function will interpolate. For the given data points we learn their latent distributions from our encoder and we sample $\mathbf{z}$ and $\mathbf{z}'$. Then we can plug in those two points in the equation of linear interpolation that is given below:

$$\mathbf{z}_\lambda = \lambda \mathbf{z} + (1 - \lambda)\mathbf{z}'$$

Finally we can use our decoder to obtain the sample $\mathbf{x}_\lambda$ from the learned distribution $p(\mathbf{X}|\mathbf{z}_\lambda)$. This sample is by itself a reconstructed image since we have predicted-interpolated the values between the pixels of each given vector. We plot a figure containing a grid. On each row, the leftmost entry is $\mathbf{x}$, the rightmost entry is $\mathbf{x}'$, and the k entries in between are given by x_{λ_i} , with λ_i on a uniform k-partition of [0, 1]. We chose the right-hand side (RHS) as an "anchor" image and we use all the other class labels to obtain our results as shown at Figure 15. We can see that for $\lambda = 0$ we interpolate exactly the point $\mathbf{x}$ and for $\lambda = 1$ we interpolate the point $\mathbf{x}'$. For all the other lambdas we can see that the interpolate images inherit features from the 2 images with different analogies. On the RHS an image looks more to the anchor image while on the LHS we can see the reconstructed images to look like their respected class-digit.

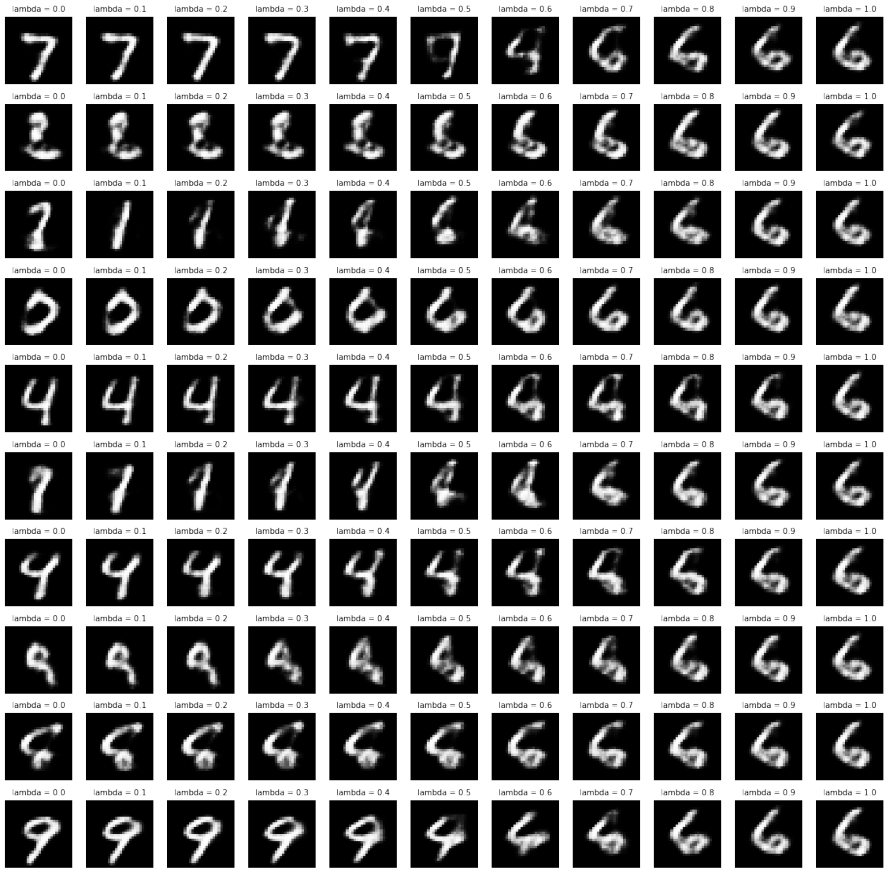

Figure 15: Reconstructed images from linear interpolation between two vectors (data points)

Task 4: Inference Without the Encoder

For this task, we will attempt to estimate the distribution $p(\mathbf{Z} \mid \mathbf{x})$ without using the encoder. Specifically, we use the Bernoulli distribution of Task 2d to construct $p(\mathbf{X} \mid \mathbf{z}$.

Task 4a: Reconstruction without the Encoder

In this task, we consider a single data point $\mathbf{x}$ from the testing data. A diagonal Gaussian $q(\mathbf{Z} \mid \Psi)$ with independent components over $\mathcal{Z}$ is used to estimate the distribution $p(\mathbf{Z} \mid \mathbf{x})$. Ψ contains the mean vector and the diagonal of the co-variance matrix. We maximize the ELBO for $\mathbf{x}$ with respect to Ψ by

$$\Psi_* = \arg\max_{\Psi} \mathbb{E}_{q(\mathbf{Z}|\Psi)}[\log p(\mathbf{x} \mid \mathbf{z})] - \mathrm{KL}(q(\mathbf{Z} \mid \Psi) \| p(\mathbf{Z}))$$

We use the Bernoulli distribution to construct $p(\mathbf{X} \mid \mathbf{z}$, also use the interpretation of the data of Task 2d.

Hence, rather than the usual expression for the ELBO, our training method instead maximize

$$-H(p(\mathbf{B} \mid \mathbf{x}), p(\mathbf{B} \mid \mathbf{z})) - \mathrm{KL}(q(\mathbf{Z} \mid \Psi) \| p(\mathbf{Z}))$$

In implementation, we set the decoder to be non-trainable and construct two trainable parameters μ and log-variance to represent the mean vector and the diagonal of the co-variance matrix of Ψ. We randomly initialized the value of μ and log-variance from a standard normalize distribution. We use stochastic gradient descent to optimize μ and log-variance with learning rate $= 0.001$. To approximately evaluate $\mathbb{E}_{q(\mathbf{Z}|\Psi)}[\log p(\mathbf{x} \mid \mathbf{z})]$, we use one-sample Monte Carlo estimate and the reparameterization trick to estimate the value of $-H(p(\mathbf{B} \mid \mathbf{x}), p(\mathbf{B} \mid \mathbf{z}))$.

For a single data point $\mathbf{x}$, we train the parameters μ and log-variance for 500 epochs to ensure convergence. We also use the early stopping mechanism described in Task 1c with the negative ELBO value as the indicator. In this task we set *patience=50*.

We randomly choose 10 data points from the testing data, the reconstruction of them are shown in Figure 16.

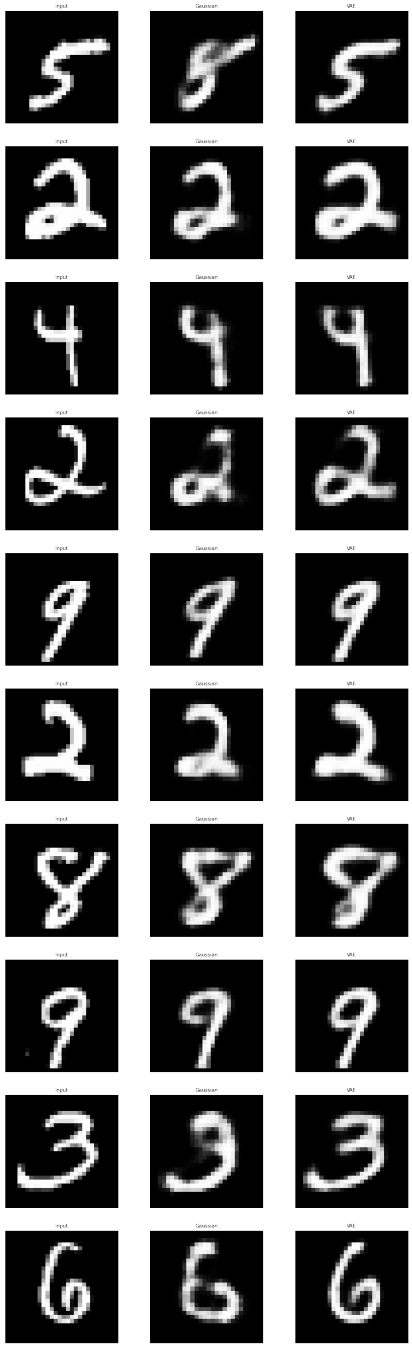

Figure 16: Original data point (leftmost column). Reconstructions without encoder (mid column). Reconstructions with encoder (rightmost column)

Task 4b: Image Completion

In this task, we use the techniques from Task 4a to perform image completion. We also consider a single data point $\mathbf{x}$ from the testing data, and then consider the partition of the variables $\mathbf{X}$ into parts $\mathbf{X}_L$ and $\mathbf{X}_R$, representing respectively the left- and right half of the image. Our goal will be to construct a complete image x' given only the left half x_L; that is, we want to reconstruct x'_R. We maximize the ELBO only for $\mathbf{x}_L$ with respect to Ψ by computing,

$$\Psi_* = \arg\max_{\Psi} \mathbb{E}_{q(\mathbf{Z}|\Psi)} \left[\log p\left(\mathbf{x}_L \mid \mathbf{z}\right)\right] - \mathrm{KL}(q(\mathbf{Z} \mid \Psi) \| p(\mathbf{Z}))$$

After the convergence of the optimization of Ψ, we generate a sample $\mathbf{z}$ from $q\left(\mathbf{Z} \mid \Psi_*\right)$ and then reconstruct the right half of the input image by samling $\mathbf{x'}_R$ from $p\left(\mathbf{X}_R \mid \mathbf{z}\right)$.

Same as Task 4a, we also use the Bernoulli distribution to construct $p(\mathbf{X}_L \mid \mathbf{z})$. In implementation, we set the decoder to be non-trainable and construct two trainable parameters μ and log-variance to represent the mean vector and the diagonal of the co-variance matrix of Ψ. We randomly initialized the value of μ and log-variance from a standard normalize distribution. We use stochastic gradient descent to optimize μ and log-variance with learning rate = 0.001.

We randomly choose 10 data points from the testing data, the reconstruction of them are shown in Figure 17.

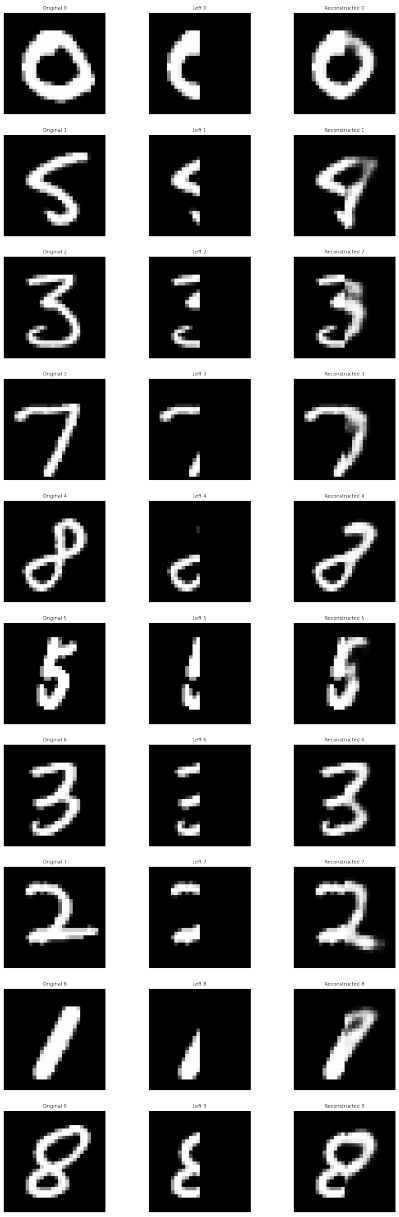

Figure 17: Original data point (leftmost column). Left part of original data points (mid column). Completed image (rightmost column)

Peer Review

Equal contribution. Yang did question 1,2,4. Iason did 1,2,3. Then we corrected and revisited all the missing parts evenly.