THE ANSWER SHEET FOR THE FINAL PROJECT, PROBABILISTIC MACHINE LEARNING

A PREPRINT

Minhao Zhang
id:cpg369

Siyi Qi
id:rwx433

April 11, 2024

A Density Model

A.1 Improved output distribution

Deliverable 1

The probability mass function f of a Bernoulli distribution is

$$f(k;p) = p^k(1-p)^{1-k} \text{ for } k \in \{0,1\}$$

In a Variational Autoencoder (VAE), the reconstruction loss can be written as $-\log p_\theta(x|z)$, so the Bernoulli case can be written as:

$$\text{Cross Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}[k_i \log(p_i) + (1-k_i)\log(1-p_i)]$$

It is a binary cross entropy loss. Hence, a Bernoulli output distribution in a VAE leads to a binary cross entropy loss.

Deliverable 2

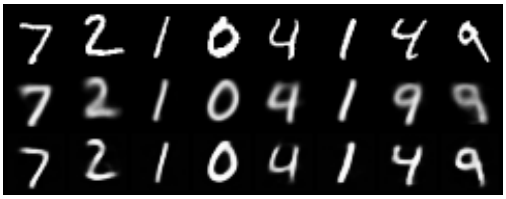
As the loss function is equivalent to a Binary Cross Entropy, we are assuming a Bernoulli distribution of the data. This means that it assumes our data is either 0 or 1. However, our MNIST data is in the range of $[0, 255]$ where we re-scaled into $[0, 1]$. If we actually draw samples from the output distribution, we only get 0 or 1 values. Hence, this is clearly wrong mathematically. To overcome this, people used the mean of the Bernoulli distribution as the actual reconstruction instead of sampling from the distribution to form a reconstruction. This, surprisingly, gave decent results.

Deliverable 3

The reason why we choose the newly-proposed Continuous Bernoulli is that it provides a continuous mapping which could be used to model a almost continuous MNIST dataset. To do this, we changed the decoder of our model to output the mean of the Continuous Bernoulli. Here are the relevant code.

```
def decode(self, z: torch.Tensor):
    h3 = F.dropout(F.relu(self.dec_h(z)), p=0.1)
    h4 = F.dropout(F.relu(self.dec_layer(h3)), p=0.1)
    temp = torch.sigmoid(self.out_layer(h4))
    temp = ContinuousBernoulli(probs=temp)
    return temp.mean
```

In our experiment, we used an encoder of two 500 neurons layers with relu activation function and dropout of probability 0.1. Then, we use a 20-dimensional latent variable for CB-VAE and 2-dimension for B-VAE. See Appendix C.2 for more detail. Following that, the decoder is another two 500 neurons layers with relu activation and dropout of probability 0.1 and we trained 100 epochs in total. Reconstruction of the image is done using the raw parameter in the following figures, see Appendix C.3 for details.



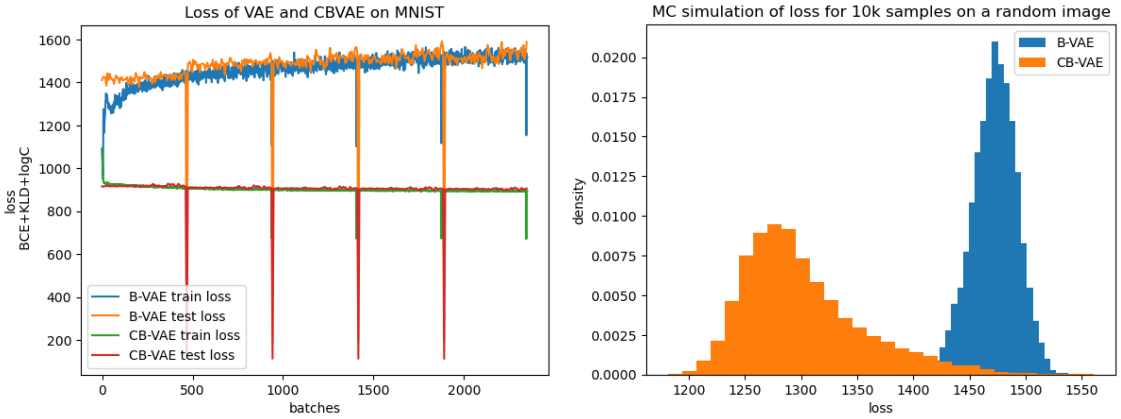
(a) Reconstruction, top to bottom: original, B-VAE, CB-VAE

(b) Sampling, top to bottom, two rows in a group: B-VAE, CB-VAE

Figure 1: VAE results comparison

By looking at the Figure 1a, we can see that the the B-VAE reconstruction is a bit blurry compared to our CB-VAE. In addition, if we pay attention to two 4s, we can see that B-VAE's reconstruction made them look like 9 instead of 4. If we look at the sampling quality of Figure 1b, we still have the blurriness in the B-VAE. There are also some digits, like the 5th on the first row and last on the second row, that are unrecognizable or a blend of two different digits. However, the bottom two rows are much sharper compared to the top two rows. Almost all digits are easily recognizable with exception of the last on the last row. Overall, we believe our CB-VAE's result is better visually.

We can also look at the CB-ELBO score during the training.



(a) ELBO comparison

(b) MC simulation comparison

Figure 2: VAE results comparison

Ignoring the dips due to change of epoch, we can see that the CB-VAE's score decreased in the first epoch and stayed about the same for the future epochs. However, the B-VAE's loss increased as we kept training. We can also see a lot of fluctuations due to the fact that it is not optimized with this particular ELBO.

On the right of the ELBO, we created a Monte Carlo approximation of $p(x)$. I used one of the test image and computed its latent space with regards to two VAEs. Then, I sampled from the latent space for 10k times and decoded the image and calculated the loss between reconstruction and original. We could obtain their expectation to be $p_{\text{B-VAE}}(x) = 1472$ and $p_{\text{CB-VAE}}(x) = 1301$. Comparing the raw value between them, we can see that our CB-VAE model is better.

A.2 Alternative density models

Deliverable 1

For the alternative density model, we used the Beta distribution. The rationale behind this choice is that Loaiza-Ganem pointed out some flaws of Beta distribution in his paper, but he did not mention the reasoning quite well [LC19]. This is also mentioned in the review of the article, the Beta distribution is more flexible, but it is more likely to overfit [Ano].

For the loss function of Beta Distribution, it is different from the previous B-VAE and CB-VAE. Taking the logarithm of the Beta distribution, we obtain its $\log p(x|z)$ to be

$$\log p(x|z) = (\alpha_z - 1)\log x + (\beta_z - 1)\log(1-x) - \log B(\alpha_z, \beta_z) \tag{1}$$

Combined with a Kullback–Leibler divergence regularization term, we obtain the Beta-ELBO of

$$\mathcal{L} = -\mathbb{E}_{q(z|x)}[\log p(x|z)] + D_{\mathrm{KL}}[q(z|x)||p(z)] \tag{2}$$

With the addition of another parameter, we changed our model output to be 2×784 where we have 784 α and 784 β. Thus, the decoder layer is now [LC19]

```
def decode(self, z: torch.Tensor) -> torch.Tensor:
    h3 = F.dropout(F.relu(self.dec_h(z)), p=0.1)
    h4 = F.dropout(F.relu(self.dec_layer(h3)), p=0.1)
    beta_params = self.out_layer(h4)
    alphas = 1e-6 + F.softplus(beta_params[:, :self.data_dim])
    betas = 1e-6 + F.softplus(beta_params[:, self.data_dim:])
    return alphas, betas
```

and the loss function is [LC19]

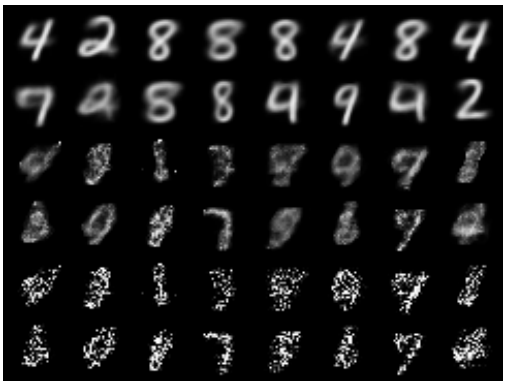
```
def beta_loss_function(alphas, betas, x, mu, logvar, beta_reg):
    x = x.view(-1, 784)
    clipped_x = torch.clamp(x, 1e-4, 1 - 1e-4)
    log_norm_const = torch.lgamma(alphas + betas) - torch.lgamma(alphas) - torch.lgamma(betas)
    log_p_all = torch.sum((alphas - 1.0) * torch.log(clipped_x) + \
        (betas - 1.0) * torch.log(1.0 - clipped_x) + log_norm_const, 1)
    log_p = torch.mean(log_p_all)
    std = torch.exp(0.5*logvar)
    KL = 0.5 * torch.sum(torch.square(mu) + torch.square(std) - torch.log(1e-8 + torch.square(st
    KL = torch.mean(KL)
    return log_p + beta_reg * KL
```

In addition to these changes, we decided to include both the mean of beta and samples drawn from Beta distributions. We formulate many Beta distributions with the parameters we decoded out and draw samples from that distribution.

Deliverable 2

After we ran the model, we obtain results like these.

(a) Reconstruction, top to bottom: original, B-VAE, Beta-VAE mean, Beta-VAE sampling



(b) Sampling, top to bottom, two rows in a group: B-VAE, Beta-VAE mean, beta-VAE sampling

Figure 3: VAE results comparison

By looking at the model, it is not hard to see that our Beta-VAE under performs the B-VAE. From the reconstruction in Figure 3a, we can see that Beta-VAE all have a mosaic-like appearance. This is due to the fact that we actually trained the model on Beta distribution instead of the reconstruction. In the sampling in Figure 3b, we see that though we could recognize some digits, most of the results are unrecognizable.

Even though our experiment with Beta-distribution VAE is not successful, we propose a potential problem where our neural network structure needs to be changed. It might also be that Beta distribution, like the Loaiza-Ganem said, overfits easily. This could be the reason behind of our relatively decent reconstruction and bad sampling.

Alongside this Beta-distribution VAE, we have implemented another VAE using the beta distribution structure with loss function of the Bernouli VAE. We do not understand the mathematical meaning behind it, but it did perform pretty well. See Appendix C.4 for details.

B Function fitting with Constraints

B.1 Fitting a standard GP

Deliverable 1
The probabilistic model $p(\theta, y|X)$ means the joint probability distribution p over the parameters of the model θ and the latent variable y, which is conditioned on the observed data X.

The function we need to learn: $g(x) = -(\sin(6\pi x))^2 + 6x^2 - 5x^4 + \frac{3}{2}$ is a periodic function with linear changes of the overall trend. We believed that a periodic kernel or the sum of a periodic kernel and a linear kernel is fine. Considering about the computational efficiency (especially for MCMC), we chose a periodic kernel $k(x, z) = \sigma^2 \exp\left(-2 \times \frac{\sin^2(\pi(x-z)/p)}{l^2}\right)$ for the GP.

We selected log-normal distributions as priors for the kernel's length-scale and variance, given that both require positive values. We adjusted the mean and standard deviation of the prior distributions to align with the characteristics of $g(x)$.

```
kernel.lengthscale = pyro.nn.PyroSample(pdist.LogNormal(0.15, 1))
kernel.variance = pyro.nn.PyroSample(pdist.LogNormal(0.05, 0.3))
```

Deliverable 2
The parameters of the MCMC were set as:

```
mcmc = pyro.infer.MCMC(nuts_kernel, num_samples=2000, num_chains=2, warmup_steps=500)
```

We did a MCMC sampling, and performed a Arviz diagnostic.

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	\
kernel.lengthscale	0.419	0.037	0.352	0.488	0.001	0.001	
kernel.variance	1.814	0.393	1.115	2.565	0.009	0.006	

	ess_bulk	ess_tail	r_hat
kernel.lengthscale	2573.0	2297.0	1.0
kernel.variance	2210.0	2269.0	1.0

Figure 4: Summary of the Arviz diagnostic

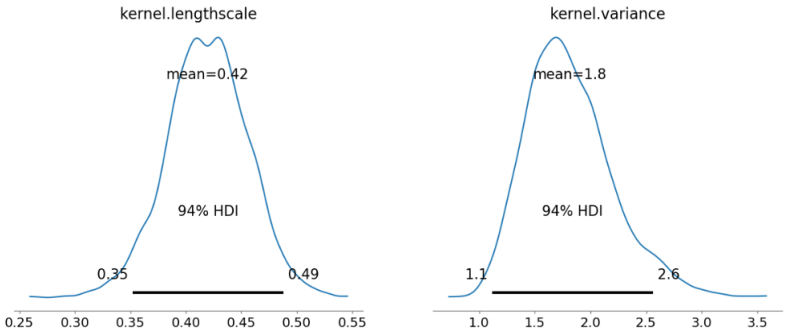


Figure 5: Posterior probability distribution of the kernel.lengthscale and kernel.variance

Figure 4 shows that the ess_bulk and ess_tail are close to the num_samples and the r_hat are 1. It indicates that the processing has a good convergence and it is more likely an efficient sampling. Figure 5 shows reasonable posterior probability distributions. Figure 6 presents a good fitting at the beginning.

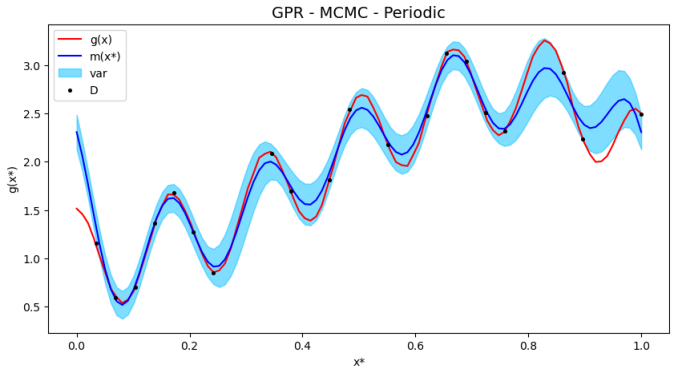


Figure 6: The fitting result with MCMC

Deliverable 3

According to the Bayes'rule, the posterior of the GP (f is the prediction) can be written as [Gul+20]:

$$p(f|X,y,\theta) = \frac{p(f|X,\theta)p(y|X,f,\theta)}{p(y|X,\theta)}$$

The prediction f* for a novel point x* is computed as [Gul+20]:

$$p(f^*|y,X,x^*,\theta) = N\Big(k(x^*,X)(K(X,X)+\sigma^2 I_N)^{-1}y, k(x^*,x^*) - k(x^*,X)(K(X,X)+\sigma^2 I_N)^{-1}[k(x^*,X)]^T\Big)$$

In addition, the likelihood function in given by:

$$p(y|S,\eta,\sigma_y^2) = N(y;0,\sigma_y^2 I_l + K(S|\eta))$$

$$\log p(y|S,\eta,\sigma_y^2) = -\frac{1}{2}y^T(\sigma_y^2 I_l + K\eta(S))^{-1}y - \frac{1}{2}\log\det(\sigma_y^2 I_l + K_\eta(S)) - \frac{l}{2}\log\sqrt{2\pi}$$

Deliverable 4

The test likelihoods with MAP: the mean is -66.6, the std is 30.89. The test likelihoods with MCMC: the mean is -2.64, the std is 2.57. The model used MCMC performed much better, because of the higher mean (we directly summing the negative log predictive likelihoods, so a higher total value indicates better performance) and lower std of the test likelihoods.

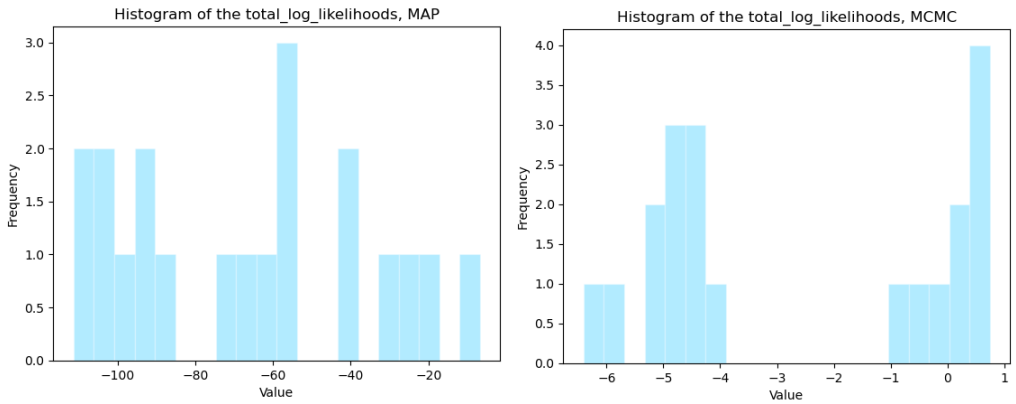


Figure 7: The test likelihoods, MAP and MCMC

B.2 Learning with Integral Constraints

Deliverable 1
We can write down the vector $(\hat{q}, f)$ as a linear transformation of f,

$$(\hat{q}, f) = (\sum_{i=1}^{l} w_i f(x_i), f(x_1), ..., f(x_l)) \tag{3}$$

According to the equivalent definitions, a random vector $X = (X_1, ..., X_k)^T$ has a multivariate normal distribution if it satisfies conditions which include "Every linear combination $Y = a_1X_1 + ... + a_kX_k$ of its components is normally distributed." In the equation (1), the first component $\sum_{i=1}^{l} w_i f(x_i)$ is a linear combination of $f(x_1), ..., f(x_l)$. Since f follows a Gaussian Process, the linear combination $f(x_1), ..., f(x_l)$ follows normal distribution. Hence, $(\hat{q}, f)$ satisfies the equivalent definitions, which means $(\hat{q}, f)$ follows a multivariate normal distribution. According to the properties of Gaussian Processes, $(\hat{q}, f)|X$ is still normal distributed. The mean of $(\hat{q}, f)|X$ is derived from the mean function of the GP. So,

$$\text{Mean}((\hat{q}, f)|X) = \begin{bmatrix} E[\hat{q}] \\ E[f(x_i)] \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{aligned} \text{Cov}((\hat{q}, f)|X) &= \begin{bmatrix} \text{Var}(\hat{q}) & \text{Cov}(\hat{q}, f(x_i)) \\ \text{Cov}(f(x_i), \hat{q}) & \text{Var}(f(x_i)) \end{bmatrix} = \begin{bmatrix} \Sigma_{\hat{q}\hat{q}} & \Sigma_{\hat{q}f} \\ \Sigma_{f\hat{q}} & \Sigma_{ff} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^{l}\sum_{j=1}^{l} w_i w_j k(x_i, x_j) & \sum_{j=1}^{l} w_1 k(x_1, x_j) & \cdots & \sum_{j=1}^{l} w_l k(x_l, x_j) \\ \sum_{j=1}^{l} w_1 k(x_1, x_j) & k(x_1, x_1) & \cdots & k(x_1, x_l) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{l} w_l k(x_l, x_j) & k(x_l, x_1) & \cdots & k(x_l, x_l) \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \text{Mean}(f|X, \hat{q}) &= \Sigma_{f\hat{q}}\Sigma_{\hat{q}\hat{q}}^{-1}\hat{q} \\ \text{Cov}(f|X, \hat{q}) &= \Sigma_{ff} - \Sigma_{f\hat{q}}\Sigma_{\hat{q}\hat{q}}^{-1}\hat{q}\Sigma_{\hat{q}f} \end{aligned}$$

Deliverable 2
The covariance matrix $\text{Cov}(f|X, \hat{q})$ has full rank if $\Sigma_{ff}, \Sigma_{\hat{q}\hat{q}}$ has full rank and $\Sigma_{ff} - \Sigma_{f\hat{q}}\Sigma_{\hat{q}\hat{q}}^{-1}\hat{q}\Sigma_{\hat{q}f}$ does not introduce liner dependencies.

Since k is an universal kernel, and the data points are distributed sparsely from 0 to 1, Σ_{ff} and $\hat{q}$ has full rank. Whats more, the process $\Sigma_{ff} - \Sigma_{f\hat{q}}\Sigma_{\hat{q}\hat{q}}^{-1}\hat{q}\Sigma_{\hat{q}f}$ avoids creating linear dependencies as it effectively eliminates the impact or contribution of $\hat{q}$ from the covariance of f. As a result, we believe that the covariance matrix $\text{Cov}(f|X, \hat{q})$ has full rank.

References

[LC19] Gabriel Loaiza-Ganem and John P. Cunningham. *The continuous Bernoulli: fixing a pervasive error in variational autoencoders*. 2019. arXiv: 1907.06845 [stat.ML].

[Gul+20] Mamikon Gulian et al. "A Survey of Constrained Gaussian Process Regression: Approaches and Implementation Challenges." In: (July 2020). URL: https://www.osti.gov/biblio/1812282.

[Adu22] Robert Aduviri. *Continuous-Bernoulli-VAE*. https://github.com/Robert-Aduviri/Continuous-Bernoulli-VAE/. 2022.

[Ano] Anonymous. *Reviews of The continuous bernoulli: Fixing a pervasive error in variational autoencoders*. URL: https://proceedings.neurips.cc/paper/2019/file/f82798ec8909d23e55679ee26bb26437-Reviews.html.

C Appendix

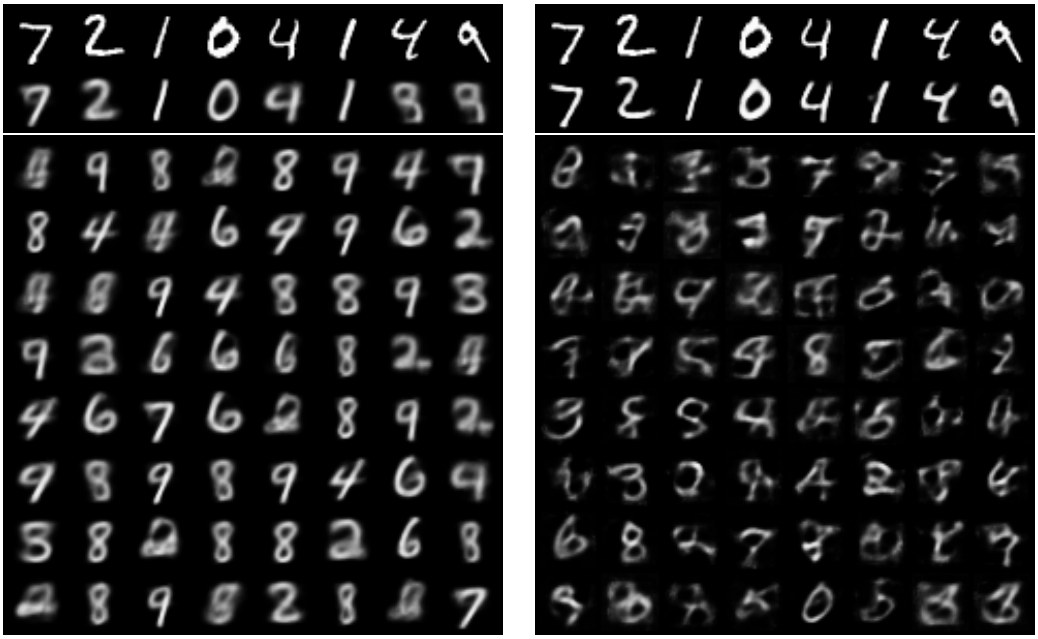
C.1 Code

Please see our code here.

C.2 Choosing 2D latent space for B-VAE

It seems like it would be best to compare models with all the same neurons layers and latent space, but we found this to be a bit hard as we found the 20-dimensional VAE over-fits very easily.

Here, I have trained two models with exact parameters except for the latent space for 10 epochs. Here are the reconstruction and samples from each model.



(a) 2D B-VAE (top: reconstruction, bottom: sample) (b) 20D B-VAE (top: reconstruction, bottom: sample)

Figure 8: B-VAE latent space comparison

If we pay attention to Figure 8a, we can see that the reconstruction is not perfect where the last two digits of 4 and 9 are blurry and they do not look very similar to the original. If we compare the reconstruction for Figure 8b, we see that every stroke of the digit is sharp and very similar to the original. However, if we look at the sampling, we clearly see that most of the result on on the right is gibberish whereas the result on left side contain many recognizable digits.

This behavior of almost perfect training and horrible testing tells us that the model with 20 dimensional latent space is overfitted. Thus, we decided to proceed with the 2 dimensional latent space model.

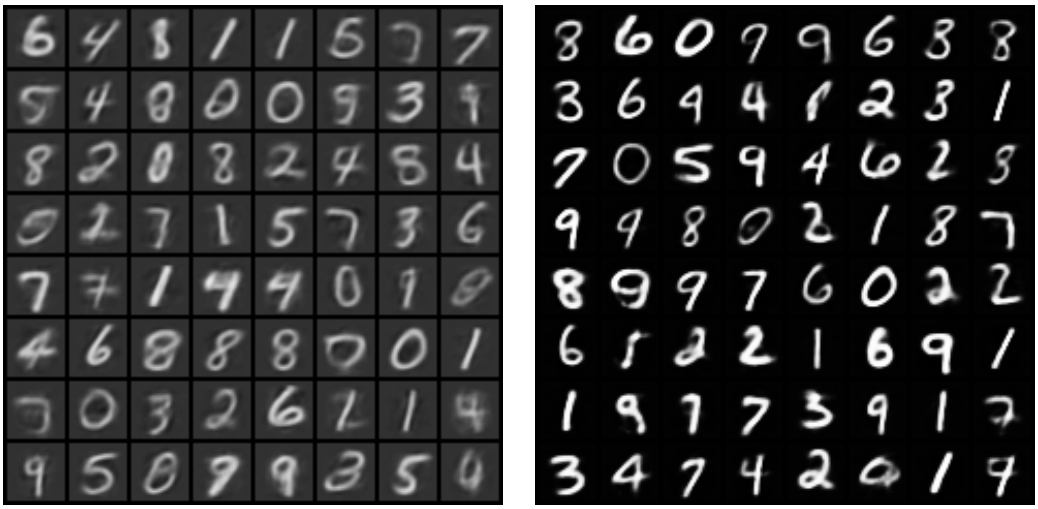
C.3 CB-VAE training and sampling

In the case of Continuous Bernoulli, we trained our model using the loss defined below [Adu22]

```
def sumlogC(x, eps=1e-5):
    x = torch.clamp(x, eps, 1.-eps)
    mask = torch.abs(x - 0.5).ge(eps)
    far = torch.masked_select(x, mask)
    close = torch.masked_select(x, ~mask)
    far_values = torch.log(
        (torch.log(1. - far) - torch.log(far)).div(1. - 2. * far))
    close_values = torch.log(torch.tensor((2.))) + \
        torch.log(1. + torch.pow(1. - 2. * close, 2)/3.)
    return far_values.sum() + close_values.sum()
```

```
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    logC = sumlogC(recon_x)
    return BCE + KLD + logC
```

With the model we defined, as `recon_x` is the mean of the Continuous Bernoulli. If we try to sample the this, we will obtain the results with a greyish background.



(a) Sampling using CB mean (b) Sampling using CB lambda

Figure 9: B-VAE latent space comparison

On the other hand, we can see the images we generated using the raw parameter λ is actually better. The reason with this behavior is that the mean of the Continuous Bernoulli cannot get to values close to 0 or 1.

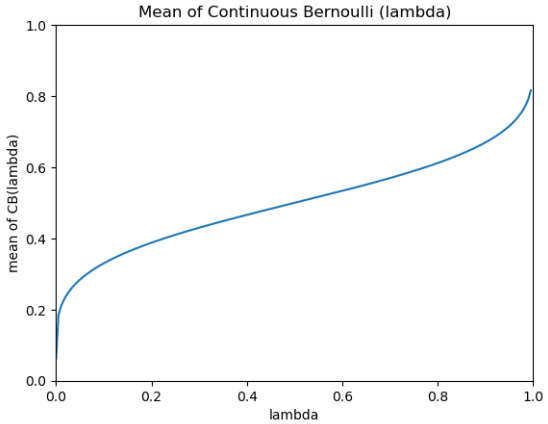
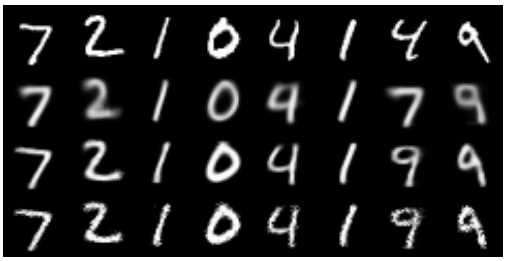


Figure 10: λ and mean of $CB(\lambda)$

If we pay attention to the left end, we see that even the parameter λ is close to 0, the value of the mean is still around 0.2. The same for the right-end where the mean seems to stop around 0.8. This results in the gray background when we try to use mean to sample. Hence, we decided to use λ as our pixel value.

C.4 Our exploratory VAE

During our trials and errors stage of writing VAE, we managed to form a VAE with decent results. We use the same VAE structure as defined in Beta-VAE. However, instead of using the proper loss function for Beta distribution, we used the decoded parameters α and β to re-construct the image using $\frac{\alpha}{\alpha+\beta}$ and used the Bernoulli-ELBO with Binary Cross Entropy and Kullback–Leibler divergence. When we actually samples, we also use the equation above to calculate the actual pixel value. To compare the result with Beta-VAE with sampling, we also included result usingn learned α and β to sample the actaul pixel. Let's call this VAE n-VAE.



(a) Reconstruction, top to bottom: original, B-VAE, n-VAE mean, n-VAE sampling

(b) Sampling, top to bottom, two rows in a group: B-VAE, n-VAE mean, n-VAE sampling

Figure 11: VAE results comparison

As we can see from Figure 11a, we can see that our n-VAE mean reconstruction does look better than B-VAE. It have a sharper digit overall. However, it failed to reconstruct the second 4. If we pay attention to the sampling figure 11b, it is quite clear that our n-VAE performs better with its much sharper and clearer image. However, there are some digits are unrecognizable which makes us believe this is a bit worse than CB-VAE.

We have a guess on what this VAE actually means. As we use the mean of the Beta distribution from the learned parameters, we are using Beta distribution to learn the parameter for the Bernoulli distribution. After that, we just used the regular Bernoulli-ELBO to train the model. This, however, might not be a correct interpretation and I wish to continue research this in the future.