Due Date: April 29th 23:59, 2020
Instructions

- *Read all instructions and questions carefully before you begin.*
- *For all questions, show your work!*
- *The repository for this homework is* https://github.com/CW-Huang/IFT6135H20_assignment
- ***The codes to reproduce results presented in this report can be found in Assignment-3 directory on this github link.***

Problem 1

Variational Autoencoders (VAEs) are probabilistic generative models to model data distribution $p(\boldsymbol{x})$. In this question, you will be asked to train a VAE on the *Binarised MNIST* dataset, using the negative ELBO loss as shown in class. Note that each pixel in this image dataset is binary: The pixel is either black or white, which means each datapoint (image) a collection of binary values. You have to model the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, i.e. the decoder, as a product of bernoulli distributions.[1]

1. **(`unittest`, 4 pts)** Implement the function '`log_likelihood_bernoulli`' in '`q1_solution.py`' to compute the log-likelihood $\log p(\boldsymbol{x})$ for a given binary sample $\boldsymbol{x}$ and Bernoulli distribution $p(\boldsymbol{x})$. $p(\boldsymbol{x})$ will be parameterized by the mean of the distribution $p(\boldsymbol{x} = 1)$, and this will be given as input for the function.

2. **(`unittest`, 4 pts)** Implement the function '`log_likelihood_normal`' in '`q1_solution.py`' to compute the log-likelihood $\log p(\boldsymbol{x})$ for a given float vector $\boldsymbol{x}$ and isotropic Normal distribution $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$. Note that $\boldsymbol{\mu}$ and $\log(\boldsymbol{\sigma}^2)$ will be given for Normal distributions.

3. **(`unittest`, 4 pts)** Implement the function '`log_mean_exp`' in '`q1_solution.py`' to compute the following equation[2] for each $\boldsymbol{y}_i$ in a given $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_i, \ldots \boldsymbol{y}_M\}$;

$$\log \frac{1}{K} \sum_{k=1}^{K} \exp\left(y_i^{(k)} - a_i\right) + a_i,$$

   where $a_i = \max_k y_i^{(k)}$. Note that $\boldsymbol{y}_i = [y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(k)}, \ldots, y_i^{(K)}]$s.

4. **(`unittest`, 4 pts)** Implement the function '`kl_gaussian_gaussian_analytic`' in '`q1_solution.py`' to compute KL divergence $D_{\mathrm{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\right)$ via analytic solution for given p and q. Note that p and q are multivariate normal distributions with diagonal covariance.

[1] The binarized MNIST is not interchangeable with the MNIST dataset available on `torchvision`. So the data loader as well as dataset will be provided.

[2] This is a type of log-sum-exp trick to deal with numerical underflow issues: the generation of a number that is too small to be represented in the device meant to store it. For example, probabilities of pixels of image can get really small. For more details of numerical underflow in computing log-probability, see `http://blog.smola.org/post/987977550/log-probabilities-semirings-and-floating-point`.

5. **(`unittest`, 4 pts)** Implement the function '`kl_gaussian_gaussian_mc`' in '`q1_solution.py`' to compute KL diveregence $D_{\mathrm{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})\right)$ by using Monte Carlo estimate for given p and q. Note that p and q are multivariate normal distributions with diagonal covariance.

6. **(`report`, 15 pts)** Consider a latent variable model $p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})dz$. The prior is define as $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_L)$ and $\boldsymbol{z} \in \mathbb{R}^L$. Train a VAE with a latent variable of 100-dimensions ($L = 100$). Use the provided network architecture and hyperparameters described in '`vae.ipynb`'[3]. Use ADAM with a learning rate of 3×10^{-4}, and train for 20 epochs. Evaluate the model on the validation set using the **ELBO**. Marks will neither be deducted nor awarded if you do not use the given architecture. Note that for this question you have to:

   (a) Train a model to achieve an average per-instance ELBO of ≥ -102 on the validation set, and report the ELBO of your model. The ELBO on validation is written as:

$$\frac{1}{|\mathcal{D}_{\text{valid}}|}\sum_{\boldsymbol{x}_i\in\mathcal{D}_{\text{valid}}}\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i) \geq -102$$

   Feel free to modify the above hyperparameters (except the latent variable size) to ensure it works.

```
In [28]:
for i in range(20):
    # train
    for x in train:
        optimizer.zero_grad()
        z_mean, z_logvar, x_mean = vae(x)
        loss = vae.loss(x, z_mean, z_logvar, x_mean)
        loss.backward()
        optimizer.step()

    # evaluate ELBO on the valid dataset
    with torch.no_grad():
        total_loss = 0.
        total_count = 0
        for x in valid:
            total_loss += vae.loss(x, *vae(x)) * x.size(0)
            total_count += x.size(0)
        print('-elbo: ', (total_loss / total_count).item())
```

```
-elbo:  167.1219940185547
-elbo:  142.9504852294922
-elbo:  129.03280639648438
-elbo:  122.30564880371094
-elbo:  116.72787475585938
-elbo:  113.34302520751953
-elbo:  110.86123657226562
-elbo:  108.96890258789062
-elbo:  107.67586517333984
-elbo:  106.59246063232422
-elbo:  105.75444793701172
-elbo:  104.75553131103516
-elbo:  104.35298919677734
-elbo:  104.01703643798828
-elbo:  103.03252410888672
-elbo:  102.75845336914062
-elbo:  102.19798278808594
-elbo:  102.01860046386719
-elbo:  101.64897155761719
-elbo:  101.37193298339844
```

Figure 1: ELBO evolution as training progresses

7. **(`report`, 15 pts)** Evaluate *log-likelihood* of the trained VAE models by using importance sampling, which was covered during the lecture. Use the codes described in '`vae.ipynb`'. The

[3]This file is executable in Google Colab. You can also convert `vae.ipynb` to `vae.py` using the Colab.

formula is reproduced here with additional details:

$$\log p(\boldsymbol{x} = \boldsymbol{x}_i) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(\boldsymbol{x} = \boldsymbol{x}_i | \boldsymbol{z}_i^{(k)})\ p(\boldsymbol{z} = \boldsymbol{z}_i^{(k)})}{q_\phi(\boldsymbol{z} = \boldsymbol{z}_i^{(k)} | \boldsymbol{x}_i)}; \quad \text{for all } k\text{: } \boldsymbol{z}_i^{(k)} \sim q_\phi(\boldsymbol{z} | \boldsymbol{x}_i)$$

and $\boldsymbol{x}_i \in \mathcal{D}$.

(a) Report your evaluations of the trained model on the test set using the log-likelihood estimate ($\frac{1}{N}\sum_{i=1}^{N} \log p(\boldsymbol{x}_i)$), where N is the size of the test dataset. Use $K = 200$ as the number of importance samples, D as the dimension of the input ($D = 784$ in the case of MNIST), and $L = 100$ as the dimension of the latent variable.

The log likelihood is -95.58343505859375 for the test set under given set of parameters.

In [31]:

```
total_loss = 0.
total_count = 0
with torch.no_grad():
    #x = next(iter(test))
    for x in test:
        # init
        K = 200
        M = x.size(0)

        # Sample from the posterior
        z_mean, z_logvar = vae.encode(x)
        eps = torch.randn(z_mean.size(0), K, z_mean.size(1))
        z_samples = z_mean[:, None, :] + torch.exp(z_logvar / 2.)[:, None, :] *
eps # Broadcast the noise over the mean and variance

        # Decode samples
        z_samples_flat = z_samples.view(-1, z_samples.size(-1)) # Flatten out th
e z samples
        x_mean_flat = vae.decode(z_samples_flat) # Push it through

        # Reshape images and posterior to evaluate probabilities
        x_flat = x[:, None].repeat(1, K, 1, 1, 1).reshape(M*K, -1)
        z_mean_flat = z_mean[:, None, :].expand_as(z_samples).reshape(M*K, -1)
        z_logvar_flat =  z_logvar[:, None, :].expand_as(z_samples).reshape(M*K,
-1)
        ZEROS = torch.zeros(z_mean_flat.size())

        # Calculate all the probabilities!
        log_p_x_z = log_likelihood_bernoulli(torch.sigmoid(x_mean_flat), x_flat)
.view(M, K)
        log_q_z_x = log_likelihood_normal(z_mean_flat, z_logvar_flat, z_samples_
flat).view(M, K)
        log_p_z = log_likelihood_normal(ZEROS, ZEROS, z_samples_flat).view(M, K)

        # Recombine them.
        w = log_p_x_z + log_p_z - log_q_z_x
        log_p = log_mean_exp(w)

        # Accumulate
        total_loss += log_p.sum()
        total_count += M

print('log p(x):', (total_loss / total_count).item())
```

```
log p(x): -95.58343505859375
```

Figure 2: Log likelihood of trained model on test dataset.

Problem 2

Generative Adversarial Network (GAN) enables the estimation of distributional measure between arbitrary empirical distributions. In this question, you will first implement a function to estimate the Squared Hellinger as well as one to estimate the Earth mover distance. This will allow you to look at and contrast some properties of the f-divergence [4] and the Earth-Mover distance [5].

[4]Relevant reference on f-divergence: https://arxiv.org/abs/1606.00709

[5]Relevant reference on Wasserstein GAN: https://arxiv.org/abs/1701.07875

We provide samplers [6] to generate the different distributions that you will need for this question. In the same repository, we also provide the architecture of a neural network function Critic : $\mathcal{X} \to \mathbb{R}$ s.t. $\mathcal{X} \subset \mathbb{R}^2$ in model.py. For training, you may use SGD with a learning rate of $1e-3$ and a mini batch size of 512.

1. **(`report`, 4 pts)** Provide the objective function of the Squared Hellinger in your report (See Nowozin et al. – Footnote 4).
   Let's say our critic is $f_\theta(x)$ which takes sample x as input. Here, θ represents parameters of $f(x)$. The x could either come from real data or be generated from a synthetic distribution. The square Hellinger objective for such a system would look as below,

   $$\max_\theta \mathbb{E}_{x\sim P(x)}[1 - \exp f_\theta(x)] + \mathbb{E}_{x\sim Q_\phi(x)}\left[\frac{1}{\exp f_\theta(x)} - \exp f_\theta(x)\right]$$

   Here, in the context of GAN, $P(x)$ represents the empirical data-distribution while $Q_\phi(x)$ represents a generator parameterized by ϕ.

2. **(`unittest`, 4 pts)** Implement the function '`vf_squared_hellinger`' in '`q2_solution.py`' to compute the objective function for estimating the Squared Hellinger distance. Please, consider the definition given in Nowozin. We give more instruction in the code template.

3. **(`report`, 4 pts)** In your report, provide the objective function of the Wasserstein distance and the objective function of the "*Lipschitz Penalty*" [7]
   Let's say our critic is $f_\theta(x)$ which takes sample x as input. Here, θ represents parameters of $f(x)$. The x could either come from real data or be generated from a synthetic distribution. The Wasserstein distance objective for such a system would look as below,

   $$\min_\theta \mathbb{E}_{x\sim Q_\phi(x)}[f_\theta(x)] - \mathbb{E}_{x\sim P(x)}[f_\theta(x)]$$

   Here, in the context of GAN, $P(x)$ represents the empirical data-distribution while $Q_\phi(x)$ represents a generator parameterized by ϕ.
   Similarly, for Lipschitz penalty, let there be a $t \sim U[0,1], x \sim P(x), y \sim Q_\phi(x)$. We the define distribution τ as distribution of random variable $\tilde{x} = tx + (1-t)y$. Then the Lipschitz penalty objective can be expressed as below.

   $$\min_\theta \lambda \mathbb{E}_{\tilde{x}\sim\tau}[(\max\{0, ||\nabla f(\tilde{x})||_2 - 1)^2\}]$$

   The distribution τ would actually depend on the sampling strategy. The strategy above is one sampling strategy. λ is regularization coefficient, it dictates how important would Lipschitz penalty's role is in learning the critic. The overall objective of the critic would look like below,

   $$\min_\theta \mathbb{E}_{x\sim Q_\phi(x)}[f_\theta(x)] - \mathbb{E}_{x\sim P(x)}[f_\theta(x)] + \lambda \mathbb{E}_{\tilde{x}\sim\tau}[(\max\{0, ||\nabla f(\tilde{x})||_2 - 1)^2\}]$$

4. **(`unittest`, 4 pts)** Implement the functions '`vf_wasserstein_distance`' and '`lp_reg`' in '`q2_solution.py`' to compute the objective function of the Wasserstein distance and compute the "*Lipschitz Penalty*". Consider that the norm used in the regularizer is the $L2$-norm.

[6] See the assignment repository https://github.com/CW-Huang/IFT6135H20_assignment
[7] See Section 5 of https://arxiv.org/pdf/1709.08894.pdf

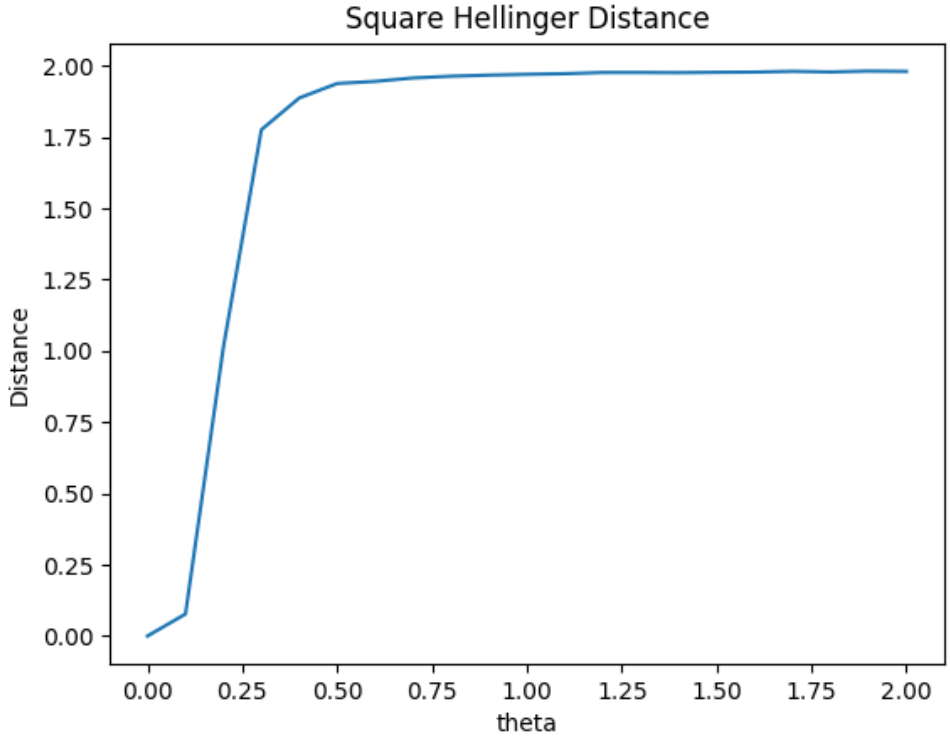

Figure 3: Plot of Squared Hellinger distance between distributions p and q.

5. **(`report`, 10 pts)** Let $u \sim U[0,1]$ be the uniform random variable with support in the interval $[0,1]$. We define p the distribution of $(0,u) \in \mathbb{R}^2$ and q the distribution of $(\theta,u) \in \mathbb{R}^2$ (We provide a function generating the p and q in the file q2_samplers.py). Plot the estimated Squared Hellinger distance between p and q and the estimated Earth-Mover distance between p and q for $\theta \in [0,2]$ with interval of 0.1 (i.e. 21 points). The x-axis should be the value of θ and the y-axis should be your estimate. In your report, provide two plots: a plot of the estimate of the Squared Hellinger with respect to θ and a plot of the estimate of the Wasserstein distance with respect to θ. Also, provide a one or two lines explanation of the behaviour you observe.

   Here, for both, Squared Hellinger and for wasserstein distance, the critic was trained for 2500 iterations with batch size of 64. Wasserstein distance is not bounded, so at θ increases, both the probability distributions are moving away from each other. Hence, Wasserstein distance increases too. While Hellinger distance is bounded between $[0,1]$. In figure 3, we are plotting twice of Hellinger distance. As *theta* increases, we can see the distance saturating to 1. The critic is able to reasonably classify as θ crosses 0.5. After that, the critic is able to distinguish between both the distributions. the critic had slight difficulties for low values of θ, which makes sense as low values of θ would make both the distributions closer to each other. The codes to reproduce this experiment can be found in this repository.

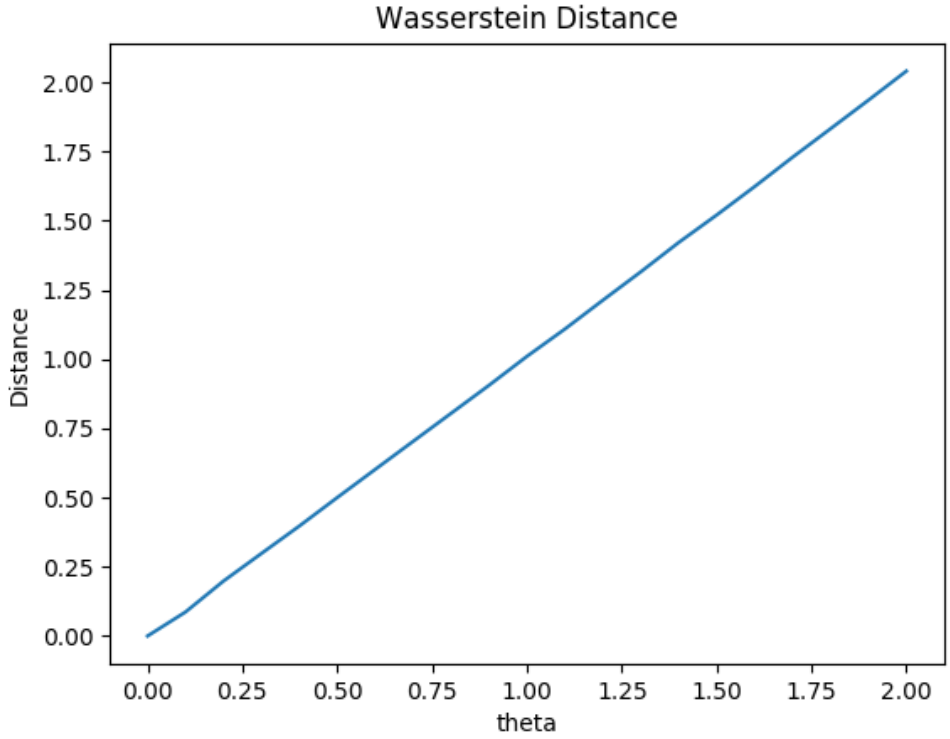

Figure 4: Plot of Earth-mover distance between distributions p and q.

Problem 3

Recent years have shown an explosion of research into using deep learning and computer vision algorithms to generate images.

In this final question, you will use the GAN framework train a generator to generate a distribution of high dimensional images $\mathcal{X} \subset \mathbb{R}^{32\times32\times3}$, namely the **Street View House Numbers** dataset (SVHN) [8]. We will consider the prior distribution $p(z) = \mathcal{N}(\mathbf{0}, I)$ the isotropic gaussian distribution. We provide a function for sampling from the SVHN datasets in 'q3_samplers.py'.

Hyperparameters & Training Pointers We provide code for the GANs network as well as the hyperparameters you should be using. We ask you to code the training procedure to train the GANs as well as the qualitative exploration that you will include in your report. You can re-use the WGAN-lp objective you wrote in the the previous question.

Qualitative Evaluation In your report,

1. **(`report`, 8 pts) Provide visual samples.** Comment the quality of the samples from each

[8] The SVHN dataset can be downloaded at http://ufldl.stanford.edu/housenumbers/

model (e.g. blurriness, diversity).

2. (`report`, **8 pts) We want to see if the model has learned a disentangled representation in the latent space.** Sample a random z from your prior distribution. Make small perturbations to your sample z for *each dimension* (e.g. for a dimension i, $z'_i = z_i + \epsilon$). ϵ has to be large enough to see some visual difference. For each dimension, observe if the changes result in visual variations (that means variations in $g(z)$). You do not have to show all dimensions, just a couple that result in interesting changes.

3. (`report`, **8 pts) Compare between interpolating in the data space and in the latent space.** Pick two random points z_0 and z_1 in the latent space sampled from the prior.

   (a) For $\alpha = 0, 0.1, 0.2 \dots 1$ compute $z'_\alpha = \alpha z_0 + (1 - \alpha) z_1$ and plot the resulting samples $x'_\alpha = g(z'_\alpha)$.

   (b) Using the data samples $x_0 = g(z_0)$ and $x_1 = g(z_1)$ and for $\alpha = 0, 0.1, 0.2 \dots 1$ plot the samples $\hat{x}_\alpha = \alpha x_0 + (1 - \alpha) x_1$.

   Explain the difference with the two schemes to interpolate between images.

   When we do the interpolation in latent space, the resulting random variable's underlying distribution is not same as that of original random variables. In this case, since our latent space is Gaussian, linear combination of that will remain Gaussian, but due to α, the scaling will change the mean and variance of the resulting random variable. Owing to this, the samples which we observe do not have a clear pattern like that of the original samples. This happens because of mix-up of disentangled representations.

   When we do interpolation in data space, we are interpolating images generated in the original space. Such interpolation lead us to confusing representations, where we see multiple digits/backgrounds/patterns being blended into a single representation. In the figure 8, we can see that representation of the generated samples are being interpolated. This is fundamentally different from latent space interpolation, where our random variable is coming from a different underlying distribution, which leads to combination of disentangled representations which have never been encountered before.

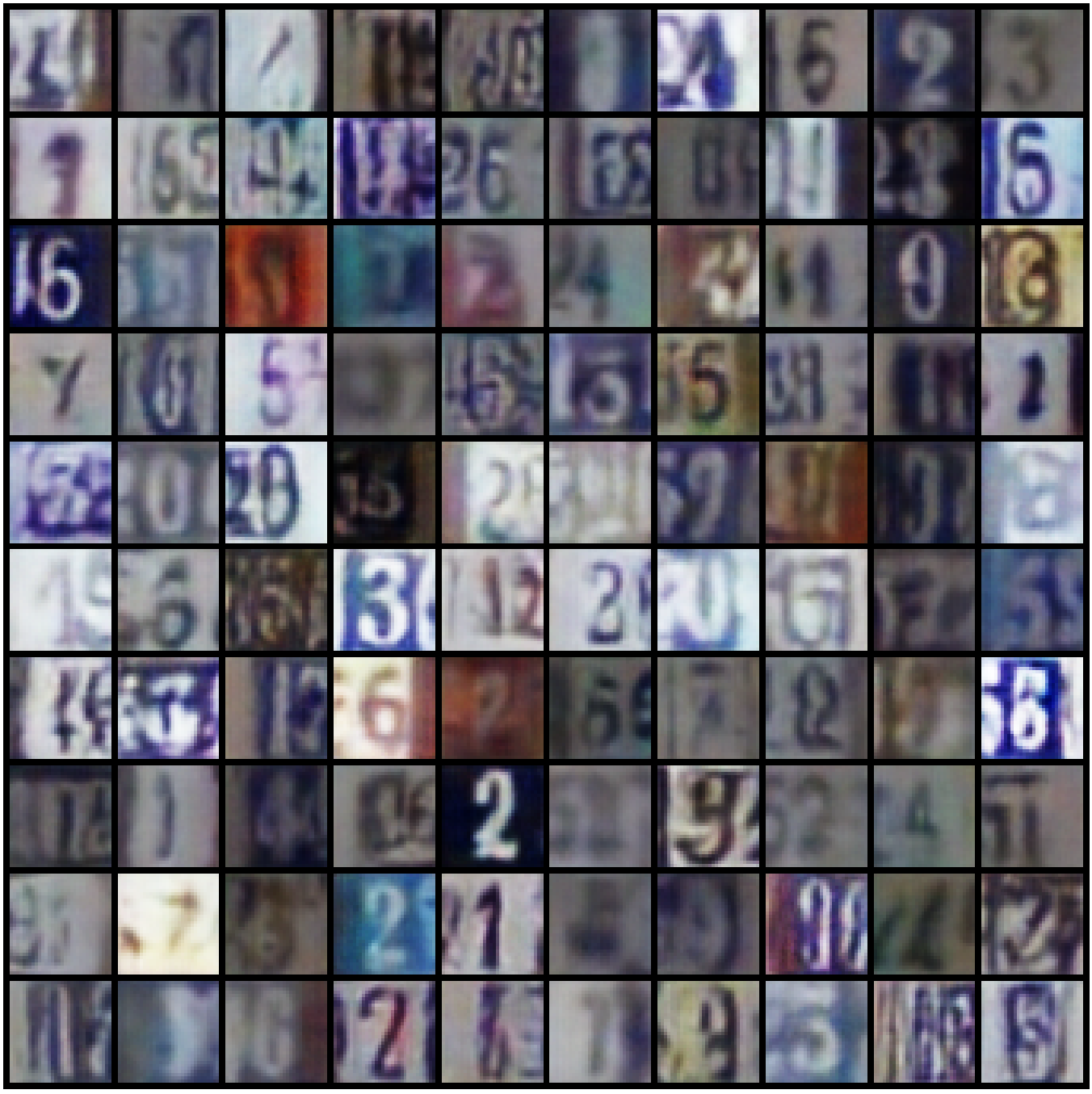

Figure 5: Question 3.1: In the image, we see samples generated after training the model for 100k iterations. The samples generated are of diverse background with varying brightness conditions. Certain samples are evidently blur and hence in poor quality. While some samples are fairly sharp in their characteristics. Like digit 2 at (8,5) has near perfect edges, same with digit 3 at (6,4). We can see the model generating noisy images at a few places like (1,1), (1,2) etc. Overall, the generated samples suggest that the model is able to generate diverse set of images with different colors, background and shape variations.

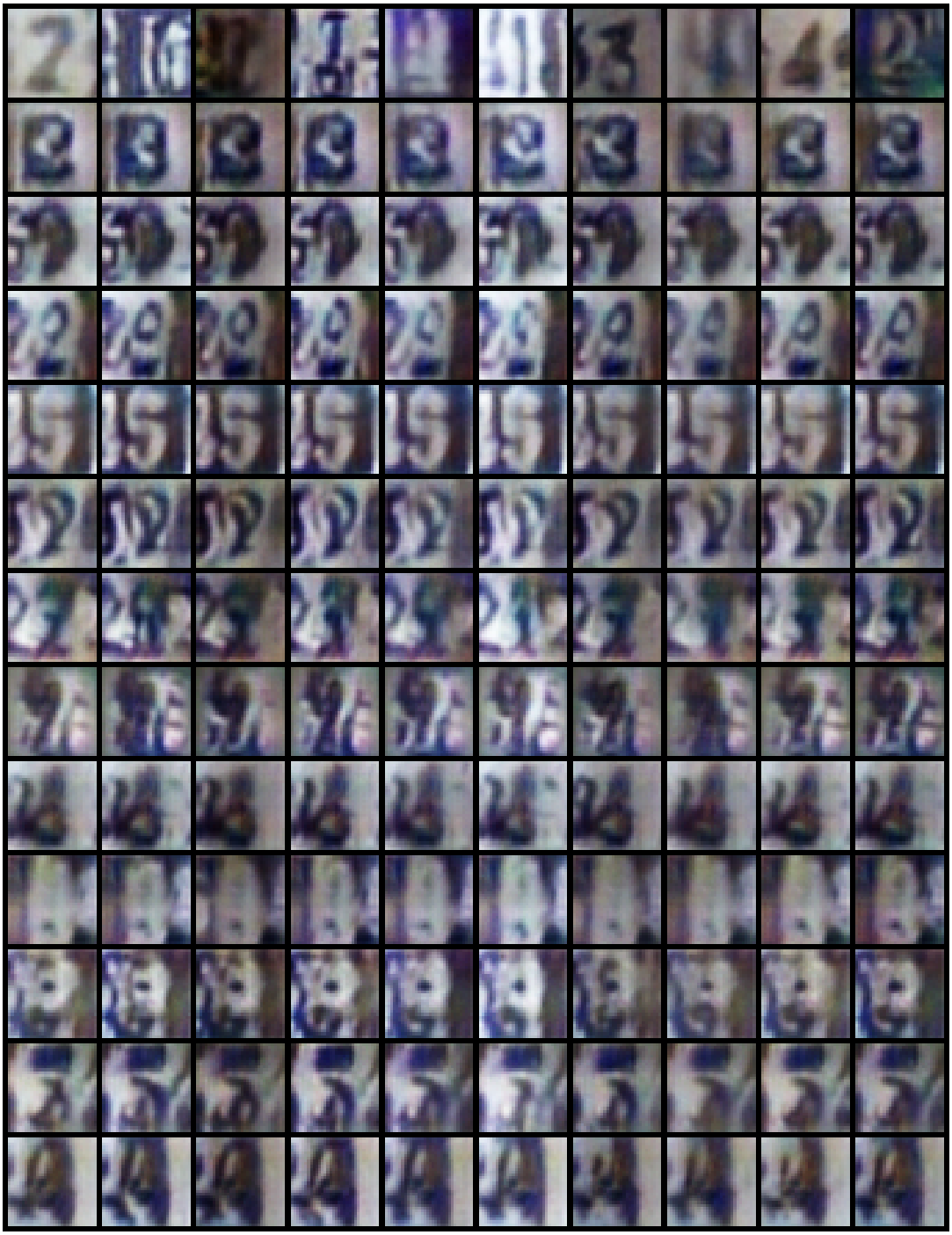

Figure 6: Question 3.2: In the figure, we show results for disentangled representation. First row corresponds to the actual images which are generated without any perturbation. Subsequent 12 rows correspond to perturbations across 12 different dimensions, 1 at a time. From row 2 to row 13, the dimensions which are perturbed are 34, 2, 72, 27, 86, 64, 80, 63, 9, 19, 41 and 10 respectively. The dimension is perturbed with $\epsilon = 50$, which is same for all dimensions. The image reflects properties of how perturbations are affecting the representations. For example, row 5 corresponds to dimension 27, and perturbing that would make the image look like 5. Similarly, row 4, which corresponds to dimension 72, has learned to represent specific structure of digit 2 with a specific background. Similarly, we can see different shapes, background colors, textures are being modeled by different dimensions. This reflects how the model has learned a disentangled representation in the latent space.

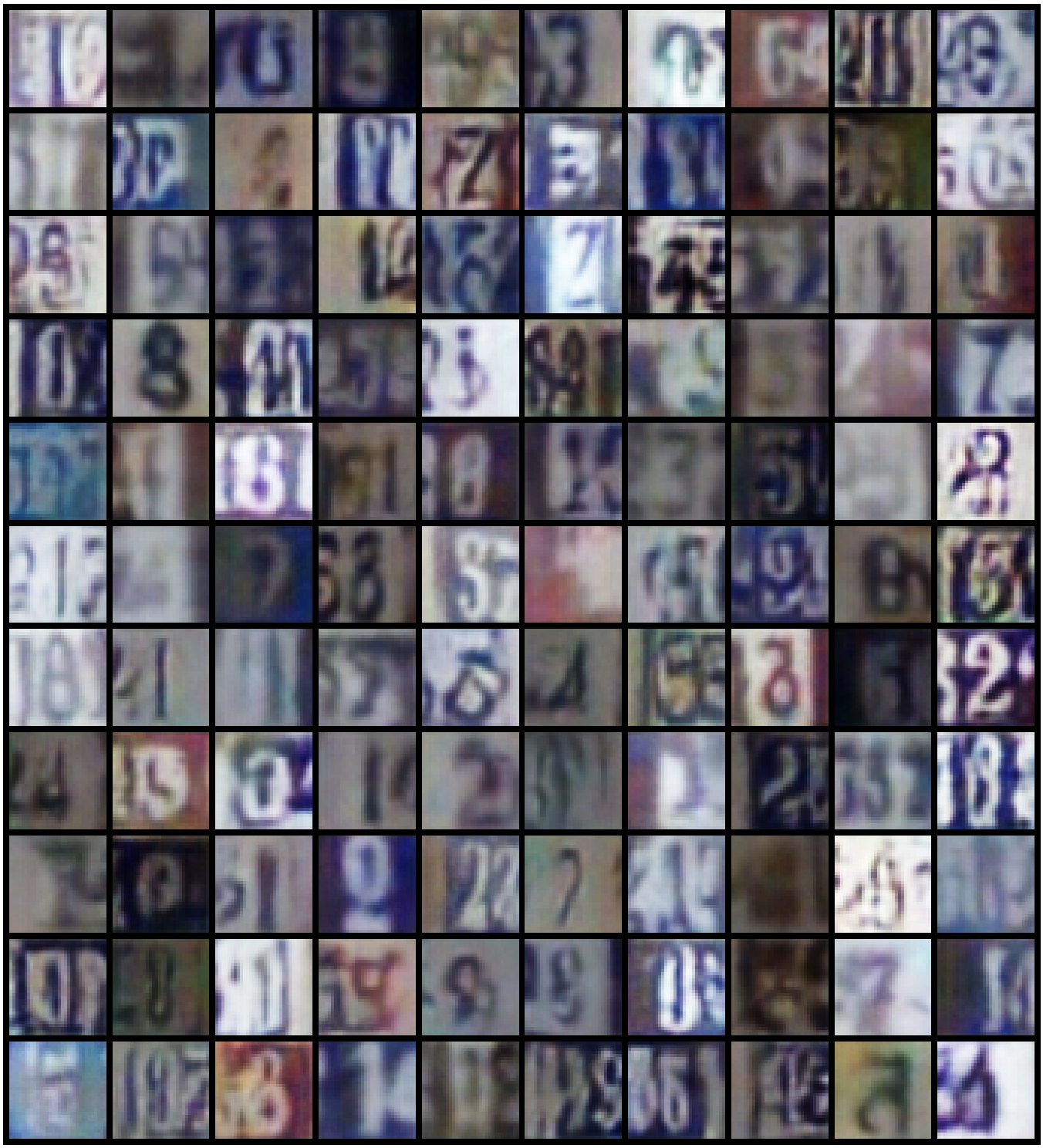

Figure 7: Question 3.3.a: here, we can see effects of image generation over interpolated latent space random variable. Row 1 to Row 11 corresponds to $\alpha \in [0, 1]$ at the interval of 0.1. we can see that as αn goes towards 0.5, the images become increasingly confusing and blurry.

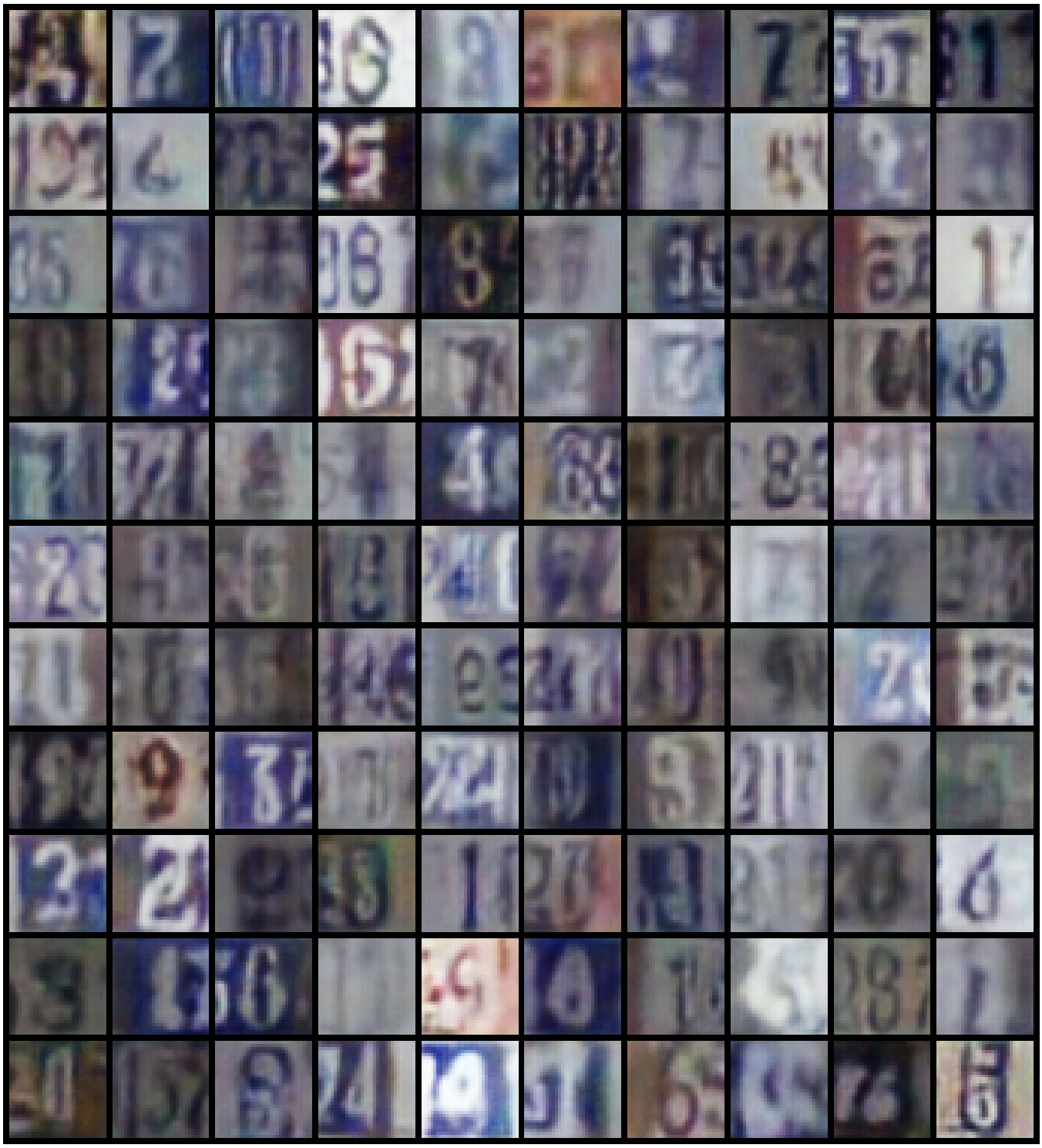

Figure 8: Question 3.3.b: When we do interpolation in data space, the generated samples look combination of individual samples. Row 1 to Row 11 corresponds to $\alpha \in [0, 1]$ at the interval of 0.1. We can see multiple digits being blended into 1 representation, and same with combination of backgrounds too.