AdvAE and FlowAE : Sampling Arbitrary Latent Variable Distributions in an Autoencoder

Srivatsan Sridhar
Stanford University
svatsan@stanford.edu

Varun Srivastava
Stanford University
vsriva@stanford.edu

Abstract

In the traditional variational autoencoder (VAE), the latent variable $\boldsymbol{z}$ is sampled from a simple prior distribution $p_\theta(\boldsymbol{z})$, which is usually Gaussian. Also, the training objective of the VAE seeks to minimize the KL-divergence between the posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and the Gaussian prior. While the simple prior makes the sampled latent variable less expressive, the training objective may impose a prior that does not fit the data distribution well. We will combine a normalizing flow model with the autoencoder in an attempt to learn an arbitrary distribution for the latent variable. For this, we propose two types of architectures. One is similar to the adversarial autoencoder (1) where a generator network will be adversarially trained to generate latent variable samples similar to an arbitrary latent space learned by an autoencoder. The second uses a normalizing flow model which will be trained to generate latent variable samples from the same arbitrary latent space.

1 Introduction

The problem being considered is high dimensional sample generation through a latent representation. It is relevant to first understand an autoencoder (AE) and a variational autoencoder (VAE) and their difference. An autoencoder consists of two networks: an encoder $\boldsymbol{z} = f_\phi(\boldsymbol{x})$ which maps an image ($\boldsymbol{x}$) to a latent variable ($\boldsymbol{z}$), and a decoder $\widehat{\boldsymbol{x}} = f_\theta(\boldsymbol{z})$ which maps the latent variable back to the same image. An AE is not a generative model but it is trained to minimize the reconstruction error between $\boldsymbol{x}$ and $\widehat{\boldsymbol{x}}$ to learn a latent representation of $\boldsymbol{x}$. On the other hand a VAE is a generative model whose encoder and decoder are parameterized probability distributions $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ respectively. It is meant to generate samples of images $\boldsymbol{x}$ through its latent representation $\boldsymbol{z}$. These two models are shown in Figure 1. In the framework of a Variational autoencoder (VAE) (2), new samples are generated by first sampling the latent variable $\boldsymbol{z}$ from a simple prior $p_\theta(\boldsymbol{z})$ such as Gaussian, and then passing it through the decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ to obtain new samples. During training, the variational inference procedure minimizes the evidence lower bound (ELBO) which in turn minimizes the KL divergence between the posterior of the latent variable $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and the Gaussian prior $p_\theta(\boldsymbol{z})$. We are exploring alternate architectures wherein we can train an autoencoder to learn an arbitrarily distributed latent space, and then sample the latent variable from this arbitrary distribution while generating new samples.

Thus, we first train an autoencoder to learn an arbitrary latent representation $\boldsymbol{z}$ of the input $\boldsymbol{x}$. In order to do this, we do not parameterize the distribution of $\boldsymbol{z}$ but rather learn a function mapping $\boldsymbol{x}$ to $\boldsymbol{z}$. The main challenge is to sample out of an arbitrary distribution directly. For this, we propose to use another network to generate samples of the latent variable. We propose two architectures which we name **AdvAE** and **FlowAE**. In the AdvAE architecture, we will use a generator network that takes a random normal vector as input and outputs a sample $\boldsymbol{z}$ in the latent space. Since we cannot

The code for this project is available at `https://github.com/VarunSrivastavaIITD/FlowVAE`

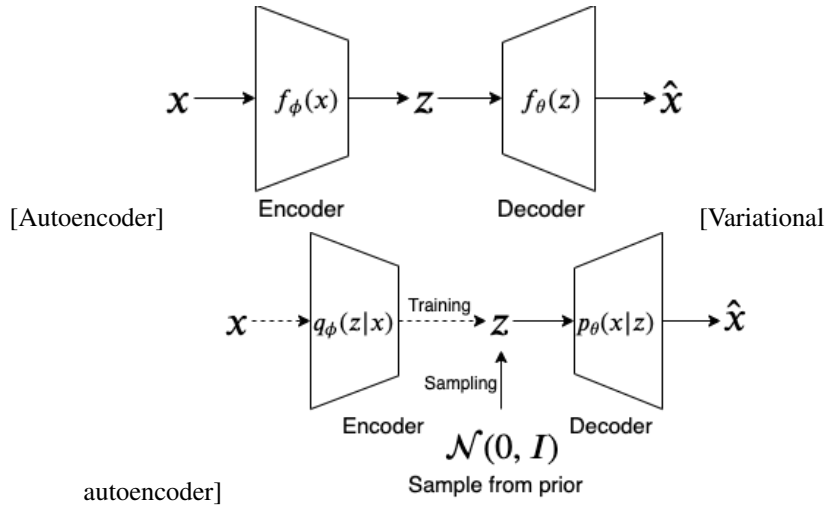


Figure 1: Comparison of AE and VAE

evaluate the likelihood of such a model, we will use adversarial training to train the generator. That is, we will use a discriminator network that classifies between $\boldsymbol{z}_{\text{fake}}$ sampled from the generator and $\boldsymbol{z}_{\text{real}}$ obtained by passing the data $\boldsymbol{x}$ through the encoder $\boldsymbol{z} = f_\phi(\boldsymbol{x})$. This method has the advantage that the generator can be any arbitrary model. In the FlowAE architecture, we use a normalizing flow model as the generator. Since we can compute exact likelihoods for this model, we train the flow model by maximizing the likelihood of $\boldsymbol{z}$ obtained from encoding real data. Although the flow model is constrained to be an invertible mapping, it is easier to train while adversarial training is usually difficult and faces problems.

We expect that the lack of a restrictive prior on $\boldsymbol{z}$ and the ability to sample it tractably will result in generation of better samples than just a VAE or a flow model. Further, we expect that the more expressive latent variable will provide better disentanglement of features. Further, this method allows us to add the ability to generate samples from any arbitrary autoencoder network that is already trained to give good reconstructions or good latent representations.

2 Related Work

Several different methods have been used to improve training of VAEs. The β-VAE (3), (4) improves disentanglement in the latent space by weighting the KL-divergence term in the evidence lower bound. The variational lossy autoencoder (5) mixes an autoregressive model with an autoencoder to learn a prior distribution on the latent space. (6) uses a normalizing flow model to enrich the posterior distribution of the latent variable.

Our work (especially the AdvAE model) is somewhat inspired from the adversarial autoencoder (AAE) in (1). In the AAE, the autoencoder is trained in two alternating phases. In the reconstruction phase, the encoder-decoder pair is trained to minimize the reconstruction error on a batch of training inputs. In the adversarial phase, the encoder and an additional discriminator network are trained on an adversarial loss (GAN-like loss). The discriminator is used to classify between latent variables generated by the encoder, and latent variables sampled from a fixed prior distribution. (The prior may be Gaussian or mixture of Gaussians or any distribution even without a closed form likelihood). Thus we may consider the AAE as an alternate training method to the VAE, using adversarial loss instead of KL divergence. Our work differs from the AAE in two aspects. Firstly, while the AAE tried to match the encoder's posterior to a fixed prior through adversarial training, we match a generator's output distribution to the encoder's posterior through adversarial training. Thus our method does not impose any prior on the latent space's distribution. Secondly, while AAE uses alternate training of the encoder-decoder and the discriminator, we fix the autoencoder after training it for a few epochs and only continue to train the generator and discriminator networks. In our results section, we also compare with results from the AAE.

While much of the previous work we have studied look at different methods to set priors for the latent variables, our method to the best of our knowledge is a novel idea to learn to sample form an arbitrary latent space distribution. (7) mentions some methods of regularizing the latent space of a non-linear autoencoder, and such methods can be used to train the autoencoder, in addition to our sampling procedures to get a better structured latent space.

3 Problem Statement

Formal Statement: To construct a learning algorithm that performs both density estimation and admits an efficient sampling scheme over a latent space $\mathcal{Z} \subset \mathbb{R}^k$.

Concretely, Given a dataset $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$ where N is the number of data points, and d is the dimensionality of each data point $\boldsymbol{x}^{(i)} \in \mathcal{X} \subset \mathbb{R}^d$ and each data point $\boldsymbol{x}^{(i)}$ sampled i.i.d. from some underlying distribution $p_{\text{data}}(\boldsymbol{x})$,

Let the autoencoder be defined by the encoder $f_\phi : \mathcal{X} \to \mathcal{Z} \subset \mathbb{R}^k$ and decoder $f_\theta : \mathcal{Z} \to \mathcal{X}$ which are trained to minimize the reconstruction error, i.e.

$$\phi, \theta = \arg\min_{\phi,\theta} \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{x}^{(i)} - (f_\theta \circ f_\phi)(\boldsymbol{x}^{(i)})||^2 \tag{1}$$

The data distribution $p_{\text{data}}(\boldsymbol{x})$ together with the encoder $f_\phi(\boldsymbol{x})$, induce a distribution $p_\phi(\boldsymbol{z})$ on the latent space. Note that it is not possible to obtain a closed form expression of this distribution because $f_\phi(\boldsymbol{z})$ is an arbitrary non-linear and non-invertible function. Then, we wish to learn a generative model $q_\psi(\boldsymbol{z})$ where $\boldsymbol{z} \in \mathcal{Z}$ which is the closest approximation to $p_\phi(\boldsymbol{z})$, i.e.

$$\psi = \arg\min_{\psi} D(p_\phi(\boldsymbol{z}), q_\psi(\boldsymbol{z})) \tag{2}$$

where D is any measure of divergence between the two distributions. By using a normalizing flow model for $q_\psi(\boldsymbol{z})$ trained using maximum likelihood, D would be the KL-divergence. By using a GAN-like loss in the adversarial training, D would be the Jenson-Shannon divergence, and by using a WGAN-like loss, D would be the Wasserstein distance. While a flow model would allow us to evaluate the density $q_\psi(\boldsymbol{z})$ for any point $\boldsymbol{z} \in \mathcal{Z}$, the adversarial network would not allow that.

Finally, we wish to sample new images $\boldsymbol{x}^* \sim p_{\text{data}}(\boldsymbol{x})$. This can be approximately done by first sampling $\boldsymbol{z}^* \sim q_\psi(\boldsymbol{z})$ and then using the decoder to get $\boldsymbol{x}^* = f_\theta(\boldsymbol{z}^*)$. We can expect $\boldsymbol{x}^*$ to follow p_{data} because $q_\psi(\boldsymbol{z})$ approximates the induced distribution $p_\phi(\boldsymbol{z})$. This can be done in both the flow model and the adversarial model because both of them allow tractable approximate sampling.

To demonstrate our model, we will generate samples from the binarized MNIST dataset, where each data point $\boldsymbol{x} \in \{0, 1\}^{784}$ ($d = 784$). We will set $k = 10$ and train the autoencoder as functions $f_\phi : \{0, 1\}^{784} \to \mathbb{R}^{10}$ and $f_\theta : \mathbb{R}^{10} \to [0, 1]^{784}$ and then round the output to convert it back to the input space. An example of a sampled MNIST image and a t-SNE visualization of the latent space is shown in Figure 2

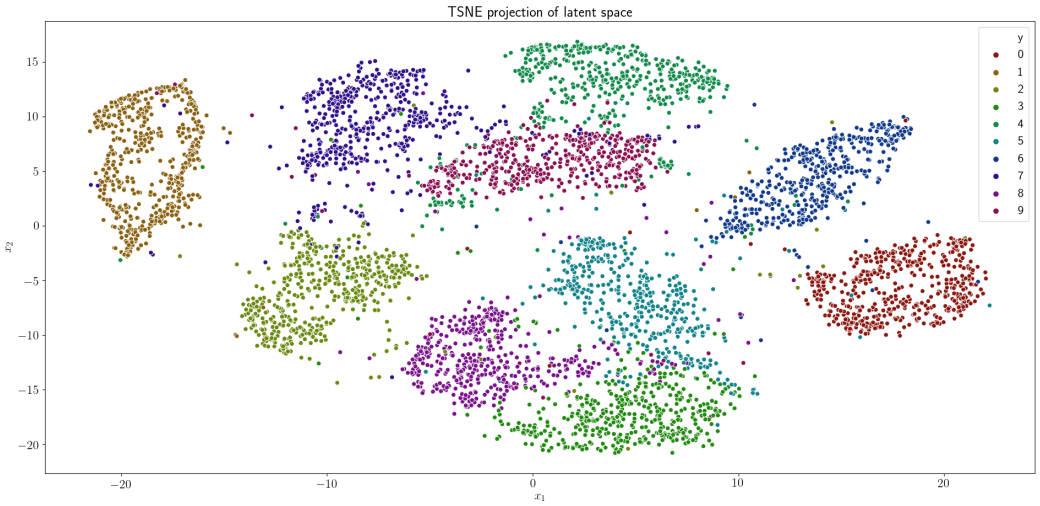
4 Approach

4.1 Autoencoder

As mentioned in Section 3, the autoencoder consists of the encoder $\boldsymbol{z} = f_\phi(\boldsymbol{x})$ and decoder $\widehat{\boldsymbol{x}} = f_\theta(\boldsymbol{z})$. Both of these are implemented using a deep neural network with 2 fully connected layers with ELU activation and batch normalization, and a linear output layer. The autoencoder is first trained using a reconstruction loss. In case of the MNIST binary images, we use binary cross entropy loss as the reconstruction loss.

$$\mathcal{L}_{AE} = \frac{1}{N} \sum_{\boldsymbol{x} \in \mathcal{D}} \text{BCELoss}(\boldsymbol{x}, \widehat{\boldsymbol{x}}) = \frac{1}{N} \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{j=1}^{784} -x_j \log(\widehat{x}_j) - (1 - x_j) \log(1 - \widehat{x}_j) \tag{3}$$

[Samples from the MNIST dataset $x \in \{0,1\}^{784}$]



[Projection of the latent vectors onto 2d space using t-SNE]

Figure 2: Illustrative instances of the dataset and the (projected) learnt latent space

4.2 FlowAE

In the FlowAE method, we use a normalizing flow model as the generator network for the latent space. The flow network that we chose to use is RealNVP (8). A flow model takes as input a random normal vector $z_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_k)$ and transforms it to an output vector $\boldsymbol{z} = F(\boldsymbol{z}_0)$. The flow model consists of 5 RealNVP layers with single layer neural networks for the scale and shift functions. The flow network is trained using maximum likelihood over the set of latent variables obtained from passing the training dataset through the encoder, i.e. $\mathcal{D}_z = \{f_\phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{D}\}$

$$\mathcal{L}_{flow} = \frac{1}{N} \sum_{\boldsymbol{z} \in \mathcal{D}_z} -\log q_\psi(\boldsymbol{z}) = \frac{1}{N} \sum_{\boldsymbol{z} \in \mathcal{D}_z} -\log \mathcal{N}(F^{-1}(\boldsymbol{z}); \mathbf{0}, \mathbf{I}_k) + \log \left| \frac{\partial F^{-1}(\boldsymbol{z})}{\partial \boldsymbol{z}} \right| \tag{4}$$

4.3 AdvAE

In the AdvAE method, we have a generator model which takes as input a random normal vector $\boldsymbol{z}_0$ and outputs $\boldsymbol{z} = G(\boldsymbol{z}_0)$. We also have a discriminator $D(\boldsymbol{z})$ which classifies between $\boldsymbol{z}$ generated from real data ($\boldsymbol{z} \sim p_\phi(\boldsymbol{z})$) and $\boldsymbol{z}$ generated by the generator ($\boldsymbol{z} \sim q_\psi(\boldsymbol{z})$). Both networks have 2 fully connected layers and an output layer. The standard non-saturating GAN loss is as follows:

$$\mathcal{L}_D = -\frac{1}{N} \sum_{\boldsymbol{x} \in \mathcal{D}} \log D(f_\phi(\boldsymbol{x})) - \mathbb{E}_{\boldsymbol{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_k)} \log(1 - D(G(\boldsymbol{z}_0))) \tag{5}$$

$$\mathcal{L}_G = -\mathbb{E}_{\boldsymbol{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_k)} \log D(G(\boldsymbol{z}_0)) \tag{6}$$

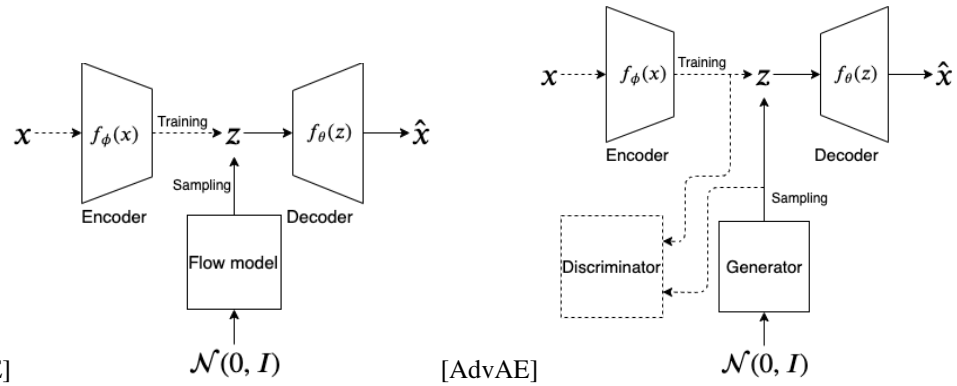


Figure 3: FlowAE and AdvAE architectures

We could also use a WGAN loss with gradient penalty, which is known to have more stable training than the standard GAN.

$$\mathcal{L}_D = -\frac{1}{N}\sum_{\boldsymbol{x}\in\mathcal{D}} D(f_\phi(\boldsymbol{x})) + \mathbb{E}_{\boldsymbol{z}_0\sim\mathcal{N}(\mathbf{0},\mathbf{I}_k)} D(G(\boldsymbol{z}_0)) + \lambda \mathbb{E}_{\boldsymbol{z}\sim r(\boldsymbol{z})}(\|\nabla D(\boldsymbol{z})\|_2 - 1)^2 \tag{7}$$

$$\mathcal{L}_G = -\mathbb{E}_{\boldsymbol{z}_0\sim\mathcal{N}(\mathbf{0},\mathbf{I}_k)} D(G(\boldsymbol{z}_0)) \tag{8}$$

The architectures of FlowAE and AdvAE are shown in Figure 3. We expect the FlowAE to give the best results in sample quality and FCD (assuming a suitable chosen and sufficiently expressive flow network), since it can learn very flexible distributions over the latent space without limiting the autoencoder's expressivity. While AdvAE admits much of the same advantages in theory, the practical difficulties in training a GAN over the latent space may hinder its performance unless extensive hyperparameter searches or effort is spent in tuning the training procedure. We expect the vanilla VAE to compare unfavourably against both these models.

5 Results

5.1 Baselines

1. **Vanilla VAE** : We compare our model with a vanilla VAE trained using the ELBO loss. The architectures for the encoder and decoder are kept the same for all the models tested. The VAE from the homework of CS 236 was used as a baseline. The prior distribution for $\boldsymbol{z}$ is standard Gaussian.
2. **AAE** : This refers to the adversarial autoencoder from (1). Again, the architectures of the autoencoder and discriminator are same as in AdvAE. In this model, the encoder-decoder are trained using cross entropy loss and the discriminator-encoder are trained using standard GAN loss. The two pairs are trained alternately. The prior distribution for $\boldsymbol{z}$ is standard Gaussian.

5.2 Experimental Procedure

Different architectures for the AdvAE and FlowAE have been implemented for generating samples from the MNIST dataset . We then compare the results of the AdvAE and FlowVAE models (against a baseline VAE and the AAE) by inspection of the generated images and by using a **Frechet Classifier Distance**.

The experimental procedure consisted of training the models (VAE, AAE, FlowAE and AdvAE) on the MNIST dataset, evaluating the sample quality at the end of training (both qualitatively using samples and quantitatively using FCD), and visualizing the latent space using t-SNE.

5.2.1 Training Scheme

The training scheme for each model is given as

1. **VAE and AAE**: The training of the autoencoder (reconstruction loss) is done concurrently with KL minimization of the imposed prior over the latent space. For the VAE, a closed form formula for the KL divergence is incorporated into the loss while the AAE uses a discriminator to accomplish the KL minimzation between some fixed prior and the latent space distribution.
2. **FlowAE and AdvAE**: The training of these models is done *sequentially*. A possible avenue of future investigation could be the instability that we observed in concurrent training of the generating network (or the flow network) with the autoencoder. Thus, in this project, we first train the autoencoder till convergence (usually 5-10 epochs) and then fix the autoencoder weights. Then, we generate samples from the latent space by encoding data from the training set and peform either MLE training (for FlowAE) or adversarial training (for AdvAE) to learn a distribution over the latent space.

5.2.2 Sampling Scheme

The sampling scheme for each model is given as:

- **VAE and AAE**: Sample $\boldsymbol{z}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$, and obtain $\boldsymbol{x}^* = f_\theta(\boldsymbol{z})$ using the decoder. The binarized image is generated by applying a sigmoid activation and rounding the output to 0 or 1 as this gives cleaner samples.
- **FlowAE** : Sample $\boldsymbol{z}_0$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$, and perform a forward pass using the **flow** network. This gives us a latent vector in the space learnt by the autoencoder $\boldsymbol{z} = F(\boldsymbol{z}_0)$. This vector is then used as an input for the decoder, and the sampling procedure is identical to the VAE thereafter.
- **AdvAE**: Sample $\boldsymbol{z}_0$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$, and perform a forward pass using the **generator** network to get $\boldsymbol{z} = G(\boldsymbol{z}_0)$. The rest of the procedure is identical to that of FlowAE.

5.2.3 Alternate architectures

The experiments for the FlowAE were performed using a RealNVP (8) flow network, which can be extended to more sophisticated flows such as Neural Spline Flows (9) and compared with baseline flow models (such as Planar, Radial and autoregressive flows). In our initial attempts, we either obtained dismal results (due to lack of sufficiently expressive output distributions in case of planar flows) or extremely expensive computation (for autoregressive flows). RealNVP flows offer the sweet spot in expressivity-speed tradeoff.

WGAN loss with gradient penalty was used for AdvAE because it gave better samples and more stable gradients than using the standard GAN loss.

Both fully connected and convolutional architectures were tested for the autoencoder and the generator in AdvAE. However we had considerable difficulties in obtaining high sample quality with the convolutional architecture (counter to intuition). Due to this, the generated samples have been shown for the fully connected case only.

5.3 Experimental Results and Metrics

The following criteria have been chosen to judge the initial performance of our models compared to the baseline VAE.

1. Frechet Classifier Distance - FID-like score where the inception net is replaced by an MNIST classifier with near-perfect accuracy $\simeq 99\%$
2. Visual Sample quality from all proposed and baseline models
3. Training dynamics for AdvAE and FlowAE
4. 2d projection of the latent space using t-SNE (10)

5.3.1 FCD comparison

Table 1 shows a comparison of FCD scores where it can be clearly observed that FlowAE offers a substantial improvement over the other models, followed by the AdvAE which is slightly inferior.

Table 1: Frechet Classifier Distance Scores

Model	FCD
Real Data	0.2
FlowAE	**4.65**
AdvAE	6.03
AAE	6.09
VAE	8.12

The baseline VAE is outperformed by all the other models. As a sanity check we also calculate the FCD for the test data (whose ideal value should be extremely low), and it can be seen that a value of 0.2 is achieved which is quite close to 0

5.3.2 Visual Quality Comparison

Figure 4 shows samples from all three models

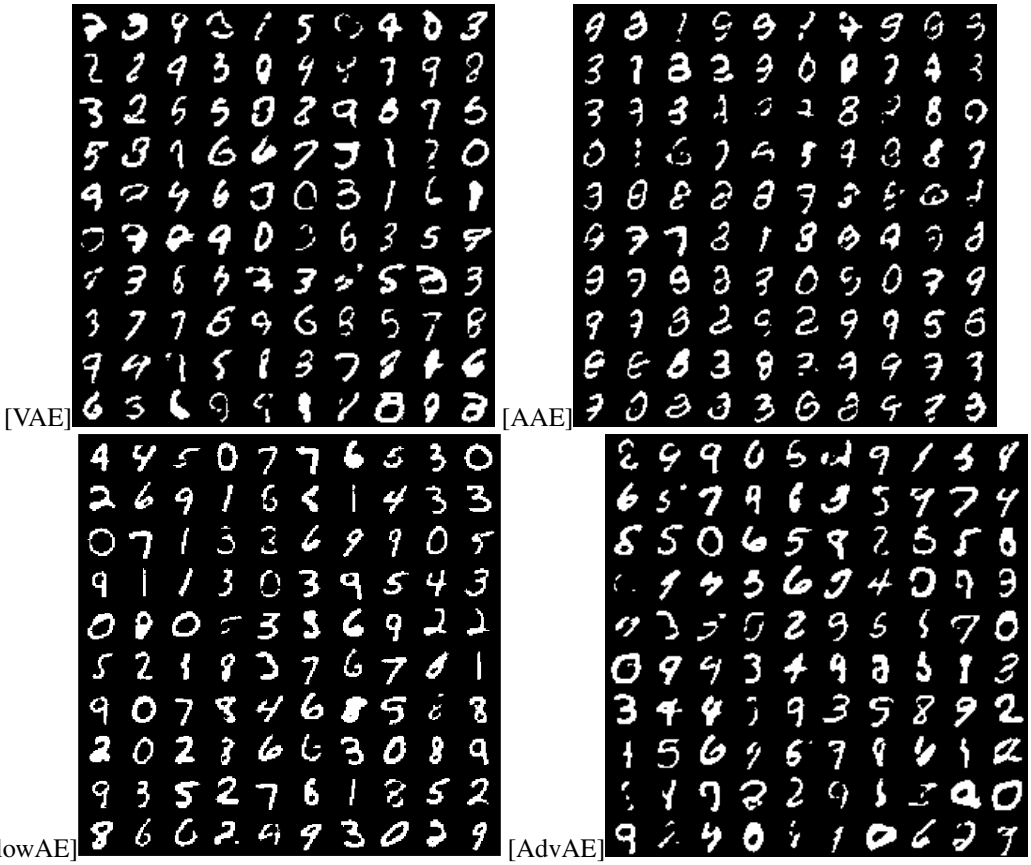


Figure 4: Samples generated from all 4 models

It can also be seen that the sampled image from FlowAE has better visual quality (which is subject to human interpretation). Due to the difficulties in adversarial training, we believe that the samples of AdvAE are not as good as those of FlowAE. While VAE samples have variety but lack clarity, AAE samples are clearer but lack variety. The FlowAE images outperform the other models in both clarity and variety.

5.3.3 Training Dynamics Comparison

Figures 5 and 6 show the training loss across the different settings for the Generative adversarial network and flow network. We can clearly see the significant advantage that the flow training provides due to both monotonic convergence, which allows for implementation of an easy stopping criterion. The instability in the GAN training means that it is difficult to know without manually inspecting samples when to stop. Note that the GAN loss is negative as a WGAN is used for training whose loss is not restricted to the positive real numbers.

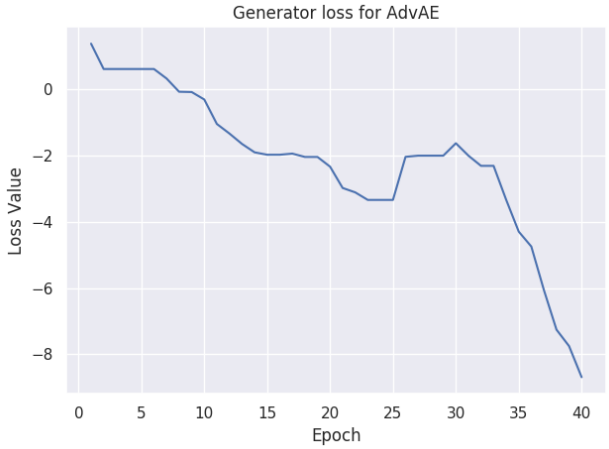


Figure 5: Training loss for the GAN

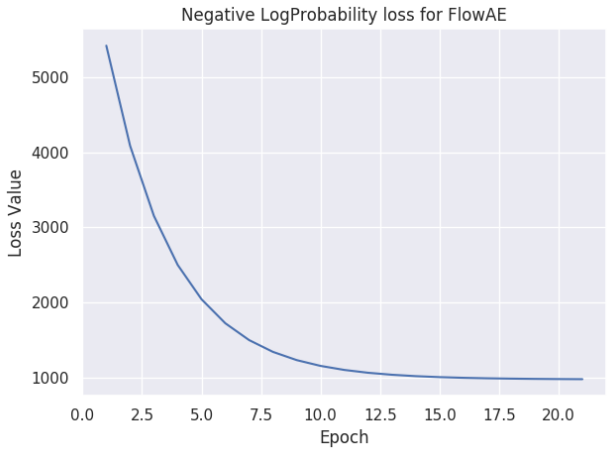
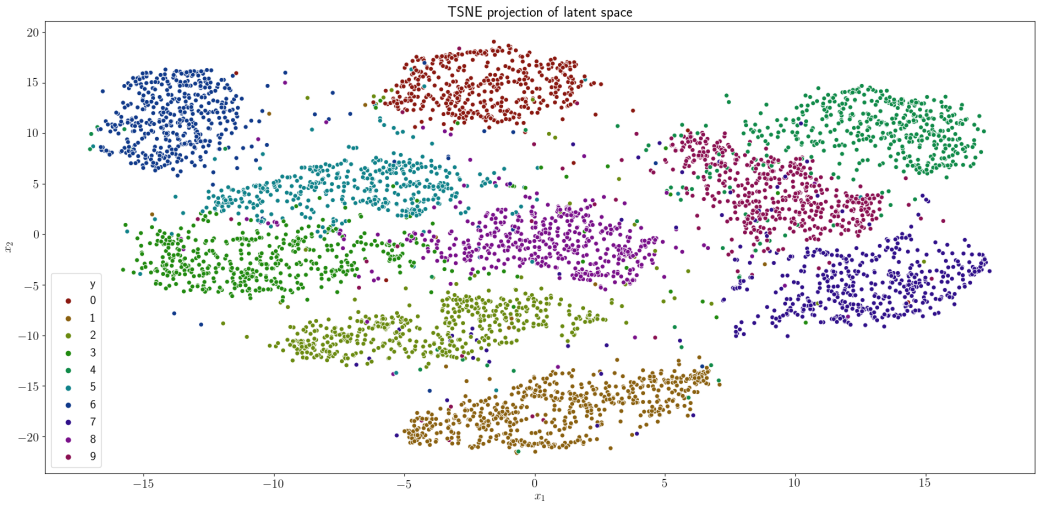


Figure 6: Training loss for the flow network

5.3.4 Latent Space Projection

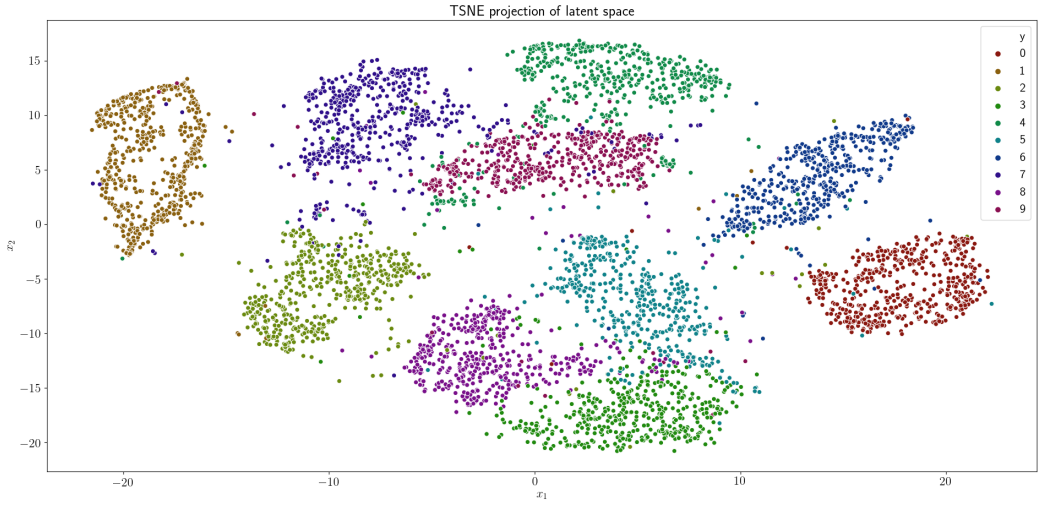
In order to better understand the characteristics of the learnt latent space, as a rudimentary comparison we project the latent vectors for the VAE and the autoencoder (which is identical for FlowAE and AdvAE) onto a 2-dimensional space using t-SNE. It can be seen in the t-SNE visualization for the autoencoder that the clusters formed are more compact than the clusters generated for VAE (for identical parameter settings and perplexity of 70 for both models).

[t-SNE for



VAE]

[t-SNE for



AE]

Figure 7: Latent space projections for VAE and AE

6 Conclusion

This project demonstrates a new method to sample a high dimensional distribution through a latent representation. Decoupling the tasks of sampling the latent space and learning a flexible class of distributions over the latent space leads to both increase in sample quality, and simplified training procedures (in FlowAE). The t-SNE plots suggest that the latent space can be clustered by its labels.

Some of the questions which we think require further investigation (which we could not do due to lack of time) will be mentioned here. Firstly, why does concurrent training of the generator (or flow) and encoder-decoder lead to instability. Secondly, is it possible to make the AdVAE perform as well as the FlowAE by architectural changes? It was interesting that while AdVAE and FlowAE could work for fully connected autoencoders, we were not able to use this framework for convolutional autoencoders (even with convolutional generators and flows). Why is the latent space of the convolutional autoencoder is much harder to learn in this framework? Is this something to do with the structure of the learnt latent space? Unfortunately, we could not extend this model to more complex datasets like CIFAR-10 due to the limitation with convolutional networks. Finally, we believe that the results of this framework could also be improved by adding to it various regularization methods developed for training autoencoders (such as in (7)).

References

[1] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[2] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[3] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *ICLR*, 2017.

[4] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in -vae," 2018.

[5] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel, "Variational lossy autoencoder," *CoRR*, vol. abs/1611.02731, 2016.

[6] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, 2015.

[7] G. Hinton, "Non-linear dimensionality reduction." `https://www.cs.toronto.edu/~hinton/csc2535/notes/lec11new.pdf`.

[8] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[9] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," *arXiv preprint arXiv:1906.04032*, 2019.

[10] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.