Deep Generative Models

Tarun Krishna
University of Amsterdam
11593040
tarun.krishna@student.uva.nl

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

How does the VAE relate to probabilistic PCA (pPCA) or Factor Analysis (FA)? You are not required to discuss the difference between pPCA and FA. Answer in 2–3 sentences.

Answer 1.1

All three statistical model are latent variable model that contains both observed and unobserved (or latent) variables. All these models look for a low dimensional manifold underlying the complex observations as contained in data. In Factor Analysis, you assume the model:

$$\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}),\ \mathbf{z} \sim \mathcal{N}(0, I)$$

Both models (FA and the VAE decoder) assume that the conditional distribution of the observable variables x on the latent variables z is Gaussian, and that the z themselves are standard Gaussians. The difference is that the decoder doesn't assume that the mean of $p(x|z)$ is linear in z, nor it assumes that the standard deviation is a constant vector. On the contrary, it models them as complex nonlinear functions of the z. In this respect, it can be seen as nonlinear Factor Analysis. PPCA is a special case of Factor Analysis, with restriction on $\Psi = \boldsymbol{\sigma}^2 I$ (assume isotropic independent variance). Factor analysis is a generalization of PCA that models non-shared variance (can think of this as noise in some situations, or individual variation in others).

1.2 Decoder: The Generative Part of the VAE

Question 1.2

Describe the procedure to sample from such a model. (Hint: ancestral sampling)

Answer 1.2

1. $\mathbf{z}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$
2. For each dimension m in $\mathbf{x}_n$ obtain the $f_\theta(\mathbf{z}_n)_m$ basically mean, by doing a forward pass of z through the decoder network.

3. Finally sample $\tilde{\mathbf{x}}_n \sim p(\mathbf{x}_n|\mathbf{z}_n)$. More precisely sample for each dimension $\tilde{x}_n^m \sim Ber(x_n^m|f_\theta(\mathbf{z}_n)_m)$

Question 1.3

This model makes a very simplistic assumption about p(Z). i.e. It assumes our latent variables follow a standard-normal distribution. Note that there is no trainable parameter in p(Z). Describe why, due to the nature of Equation 4, this is not such a restrictive assumption in practice.

Answer 1.3

As stated in Carl Doersch's tutorial [2] any distribution in d dimensions can be generated by taking a set of d variables that are normally distributed and mapping them through a sufficiently complicated function. These mapping can be learned by powerful function approximators (Neural Nets) and further we can simply learn functions which maps our independent normally distributed z values to whatever latent variables is required for the model, and then map those latent variables to X. In eq 4 $f_\theta(\mathbf{z}_n)_m$ can be thought of as mapping normally distributed z to the latent values as required by the model. And then use it further to map those latent values to a digit as stated in eq 4.

As long as we are able to use a transformation which maps our latent variable Z to any other latent variable that has a more appropriate distribution for generating our data, the choice of initial distribution does not matter for the most part. The choice of the standard normal distribution over the latent variables allows us to easily sample from it. Such transformation, albeit complex can be learned by sufficiently expressive neural networks.

Question 1.4

(a) Evaluating $\log p(\mathbf{x}_n) = \log E_{p(\mathbf{z}_n)}[p(\mathbf{x}_n|\mathbf{z}_n)]$ involves an intractable integral. However, equation 5 hints at a method for approximating it. Write down an expression for approximating $\log p(\mathbf{x}_n)$ by sampling. (Hint: you can use Monte-Carlo Integration).

(b) Although this approach can be used to approximate $\log p(\mathbf{x}_n)$, it is not used for training VAE type of models, because it is inefficient. In a few sentences, describe why it is inefficient and how does this efficiency scale with the dimensionality of z. (Hint: you may use Figure 2 in you explanation.)

Answer 1.4

(a) We can draw L samples from $p(\mathbf{z}_n)$ denoted by $\mathbf{z}_n^l$ and can approximate $\log p(\mathbf{x}_n)$ as :

$$\log p(\mathbf{x}_n) \approx \log\Big(\frac{1}{L}\sum_{l=1}^{L} p(\mathbf{x}_n|\mathbf{z}_n^l)\Big) \tag{1}$$

(b) With this approach we can estimate $\log p(\mathbf{x}_n)$ as defined in equation 1. The problem here is that in high dimensional spaces, L might be extremely large before we have an accurate estimate of $p(\mathbf{x})$. That's because for most $\mathbf{z}$, $p(\mathbf{x}|\mathbf{z})$ will be nearly zeros and hence contribute nothing to our estimate of $p(\mathbf{x})$. This can be observed in Figure 2 (assignment) i.e we have a very small overlap of densities of $p(\mathbf{z})$ and $p(\mathbf{x}|\mathbf{z})$. Well, the key idea about the variational auto-encoder is to attempt to samples values of $\mathbf{z}$ that are likely to have produced $\mathbf{x}$, and compute $p(\mathbf{x})$ from this. This means that we need a new function $q(z|X)$ (an estimate of true posterior) which can take a value of X and give us a distribution over z values that are likely to produce X.

1.3 The Encoder: $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$

Question 1.5

Assume that q and p in Equation 6, are univariate gaussians: $q = \mathcal{N}(\mu_q, \sigma_q^2)$ and $p = \mathcal{N}(0, 1)$.

(a) Give two examples of (μ_q, σ_q^2): one of which results in a very small, and one of which has a very large, KL-divergence: $\mathcal{D}_{KL}(q||p)$.

(b) Find the (closed-form) formula for$\mathcal{D}_{KL}(q||p)$.

(You do not need to derive it, you can just search for it, but feel free to derive it if that brings you joy. Hint: This will be helpful.) You will need to use this answer later.

Answer 1.5

(a) KL-Divergence between to univariate gaussian distribution is given by :

$$\mathcal{D}_{KL}(\mathcal{N}(\mu_q, \sigma_q)|\mathcal{N}(\mu_p, \sigma_p)) = \log \frac{\sigma_p}{\sigma_q} + \frac{(\mu_q - \mu_p)^2 + \sigma_q^2}{2\sigma_p^2} - \frac{1}{2}$$

We observe that for a small divergence, μ_q must be very close to μ_p and similarly for σ_q and σ_p. For a large divergence, having different mean parameters is sufficient. We know that $\mathcal{D}_{KL} \geq 0$, so setting up $\mu_q = 0, \sigma_q = 1$ and $\mu_p = 0, \sigma_p = 1$ (as given for $p = \mathcal{N}(0, 1)$) we will obtain minimum value for $\mathcal{D}_{KL}(q||p)$ which is 0. Setting $\mu_q >> 0$, $0 < \sigma_q^2 < 1$ to achieve maximum (this is one such setting).

(b) $\mathcal{D}_{KL}(q||p) = \frac{\mu_q^2 + \sigma_q^2 - 1}{2} - \log(\sigma_q)$. Considering univariate distribution.

Question 1.6

Why is the right-hand-side of Equation 11 called the lower bound on the log- probability?

Answer 1.6

Since $\mathcal{D}_{KL}(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)) \geq 0$ and $\log(p(\mathbf{x}_n))$ is fixed as a result we have that $\log p(\mathbf{x}_n) \geq E_{q(z|\mathbf{x}_n)}\big[\log p(\mathbf{x}_n|Z)\big] - \mathcal{D}_{KL}(q(Z|\mathbf{x}_n)||p(Z))$.

Question 1.7

Looking at Equation 11, why must we optimize the lower-bound, instead of optimizing the log-probability directly?

Answer 1.7

The second term on the left handside of eq 11 which is $\mathcal{D}_{KL}(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n))$ contains $p(Z|\mathbf{x}_n)$ (true posterior) which is intractable as result we cannot compute $\mathcal{D}_{KL}(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n))$. Hence we optimize the lower bound.

Question 1.8

Now, looking at the two terms on left-hand side of 11: Two things can happen when the lower bound is pushed up. Can you describe what these two things are?

Answer 1.8

Analogous to the EM algorithm, when we minimize the complexity cost $D(q(z|X)||p(z))$ with respect to the variational parameters (E-step), our lower bound increases since the KL divergence is always non-negative so we can minimize it upto 0 (an ideal scenario). Secondly we can minimize the reconstruction error $E_{z \sim q(Z|X)}[\log p(X|z)]$ (M-step) we further increase the lower bound.

However, as we may never recover the true posterior distribution given our modelling choices, the term $D(q(z|X)||p(z|X))$ is greater than zero, and so the right hand size remains a lower bound.

1.4 Specifying the encoder $\mathbf{q}_\phi(\mathbf{z}_n|\mathbf{x}_n)$

Question 1.9

Answer 1.9

The reconstruction error quantifies how well we are able to "reconstruct" the training data given the latent variables, i.e. it reflects on the quality of our learned latent variables. A good z gives a high probability $p(X|z)$ for any observation X.

The regularization term quantifies how well our latent representation matches our prior beliefs about Z. Minimizing this term moves our variational posterior to be closer to the prior $p(z)$, which corresponds to regularization and is analogous to weight decay in MAP learning.
A very nice pictorial representation was found here which is depicted below describing both reconstruction and regularization:

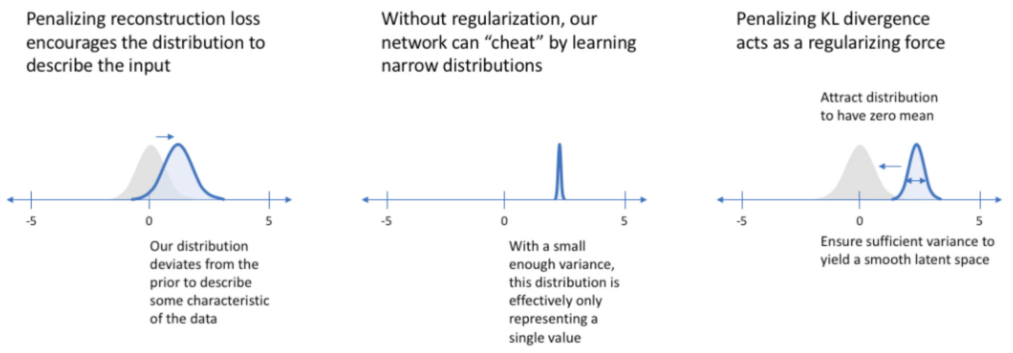

Penalizing reconstruction loss encourages the distribution to describe the input
-5
0
5
Our distribution deviates from the prior to describe some characteristic of the data
Without regularization, our network can "cheat" by learning narrow distributions
-5
0
5
With a small enough variance, this distribution is effectively only representing a single value
Penalizing KL divergence acts as a regularizing force
Attract distribution to have zero mean
-5
0
5
Ensure sufficient variance to yield a smooth latent space

Figure 1: Reconstruction and Regularization

Question 1.10

Now we have defined an objective (Equation 12) in terms of an abstract model and variational approximation, we can put everything together using our model definition (Equation 1 and 2) and definition of $q_\phi(z_n|x_n)$ (Equation 13), we can write down a single objective which we can minimize.

Write down expressions (including steps) for L_n^{recon} and L_n^{reg} such that we can minimize $L = \sum^N L_n^{recon} + L_n^{reg}$ as our final objective. Make any approximation explicit. reg

(Hint: look at your answer for question 1.5 for L_n^{reg} .)

Answer 1.10

We sample $\mathbf{z}_n^l \sim q_\phi(\mathbf{z}|X = \mathbf{x}_n)$ for $l \in 1, 2, \cdot\cdot, L$. Using $\mathbf{z}_n^l = g_\phi(\mathbf{x}_n, \epsilon^l) = \boldsymbol{\mu}_n + \boldsymbol{\sigma}_n \odot \epsilon^l$ where $\epsilon^l \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. With $\odot$ we signify an element-wise product. Also expected reconstruction $E_{q_\phi(\mathbf{z}|\mathbf{x}_n)}\big[\log p_\theta(\mathbf{x}_n|\mathbf{z})\big]$ requires sampling.

$$\begin{aligned}
\mathcal{L} &= \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n^{recon} + \mathcal{L}_n^{reg} \\
&= \frac{1}{N}\sum_{n=1}^{N} -E_{q_\phi(\mathbf{z}|\mathbf{x}_n)}\big[\log p_\theta(\mathbf{x}_n|\mathbf{z})\big] + \mathcal{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}_n)||p_\theta(\mathbf{z})) \\
&= \frac{1}{N}\sum_{n=1}^{N}\left(-\frac{1}{L}\sum_{l=1}^{L}\big(\log p_\theta(\mathbf{x}_n|\mathbf{z}_n^l)\big) + \frac{1}{2}\sum_{j=1}^{D_z}\Big(\mu_{j,n}^2 + \sigma_{j,n}^2 - 1 - \log \sigma_{j,n}^2\Big)\right)
\end{aligned}$$

D_z stands for dimension of latent variable z. One can further elaborate the term $\frac{1}{L}\sum_{l=1}^{L}\big(\log p_\theta(\mathbf{x}_n|\mathbf{z}_n^l)\big) = \frac{1}{L}\sum_{l=1}^{L}\log\sum_{i=1}^{D_x}\Big(x_n^i \log y_n^i + (1 - x_n^i)\log(1 - y_n^i)\Big)$. Where $\mathbf{y_n}$ is output from the decoder. D_x stands for dimension in inputs.

1.5 The Reparametrization Trick

Question 1.11

Read and understand Figure 4 from the tutorial by Carl Doersch. In a few sentences each, explain why:

(a) We need $\nabla_\phi \mathcal{L}$

(b) the act of sampling prevents us from computing $\nabla_\phi \mathcal{L}$

(c) What the reparametrization trick is, and how it solves this problem.

Answer 1.11

(a) Obtaining the gradient of the loss with respect to the variational parameters ϕ allows us to optimize these variables using gradient descent. Basically ϕ consists of learnable parameters for the encoder network. This allows us to use variational inference to learn a variational distribution that approximates the true posterior. This optimization can be done with gradient descent which is why we need this gradient.

(b) Sampling $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}_n)$ is a non-continous operation and has no gradient. This does not allow us to take gradients of the lower bound (which involves expectations over $q_\phi(\mathbf{z}|\mathbf{x}_n)$ with respect to the variational parameters ϕ.

(c) One often encounter problem of computing the gradient of an expectation of a smooth function f:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \nabla_\theta \int p(z;\theta) f(z) dz$$

This gradient is often difficult to compute because the integral is typically unknown and the parameters θ, with respect to which we are computing the gradient, are of the distribution $p(z;\theta)$. If we can find a transformation from samples of some random variable (which we do not wish to learn) to samples of our variational distribution (which we wish to learn), then we no longer need to sample directly from the variational distribution and this overcomes the non-differentiability problem. We require this transformation to be invertible, and furthermore it must absorb the variational parameters, i.e. be parameterized by θ such that we can take its gradients with respect to θ. Therefore, by using this transformation, we can obtain samples from the variational posterior and take gradients with respect to the parameters that govern the variational distribution. Thus this allows us to compute the gradient in a more amenable way:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))]$$

Let's derive this expression and explore the implications of it for our optimisation problem. One-liners give us a transformation from a distribution $p(\epsilon)$ to another $p(z)$, thus the differential area (mass of the distribution) is invariant under the change of variables. This property implies that:

$$p(z) = \left|\frac{d\epsilon}{dz}\right| p(\epsilon) \implies |p(z)dz| = |p(\epsilon)d\epsilon|$$

Re-expressing the troublesome stochastic optimisation problem using random variate reparameterisation, we find:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] &= \nabla_\theta \int p(z;\theta) f(z) dz \\ = \nabla_\theta \int p(\epsilon) f(z) d\epsilon &= \nabla_\theta \int p(\epsilon) f(g(\epsilon,\theta)) d\epsilon \\ = \nabla_\theta \mathbb{E}_{p(\epsilon)}[f(g(\epsilon,\theta))] &= \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))]\end{aligned}$$

In the second line, we reparameterised our random variable in terms of a one-line generating mechanism. In the final line, the gradient is now unrelated to the distribution with which we take the expectation, so easily passes through the integral. Our assumptions throughout this process were simple: 1) the use of a continuous random variable z with a known one-line reparameterisation, 2) the ability to easily generate samples ϵ from the base distribution, and 3) a differentiable function f. With this appraoch gradients flow backwards through this path, allowing for computation by automatic differentiation and composition with other gradient-based systems.

$$\mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))] = \frac{1}{S}\sum_{s=1}^{S} \nabla_\theta f(g(\epsilon^{(s)},\theta)), \quad \epsilon^{(s)} \sim p(\epsilon)$$

NOTE: Not to confuse θ with decoder parameters.

1.6 Putting things together: Building a VAE

Question 1.12

Build a Variational Autoencoder in PyTorch, and train it on the Binary MNIST data. Start with the template in a3_vae_template.py in the code directory for this assignment.

Following standard practice — and for simplicity — you may assume that the number of samples used to approximate the expectation in $\mathbf{L}_n^{recon}$ is 1.

Provide a short (no more than ten lines) description of your implementation. You will get full points on this question if your answers to the following questions indicate that you successfully trained the VAE and your description is sufficient, however, do not forget to include your code in your final submission.

Answer 1.12

(a) `VAE` class comprises of `encoder` and `decoder`.

(b) `Encoder` class serves as a inference model. Which ouput μ and σ from which $z \sim N(z|\mu, \sigma)$. This has been realised by passing input to a `nn.Linear(x)` and applying `nn.ReLU` non-linearity. Further output from this is passed on to two different affine layers returning `mean` and `std` standard deviation. For standard deviation I have used `nn.Softplus` just to make sure standard deviation is non-negative.

(c) z is sampled following reparametrization trick i.e $z \sim mean + \mathcal{N}(0, \mathbf{I}) * std$

(d) Once z is sampled it is fed into the Generative network so called Decoder or $q_\theta(\mathbf{x}|\mathbf{z})$.

(e) Decoder is realized by two `nn.Linear` layers one with `nn.ReLU` non-linearity and the last one with `nn.Sigmoid` non-linearity.

(f) Output of the Decoder is basically the reconstruction of the digits.

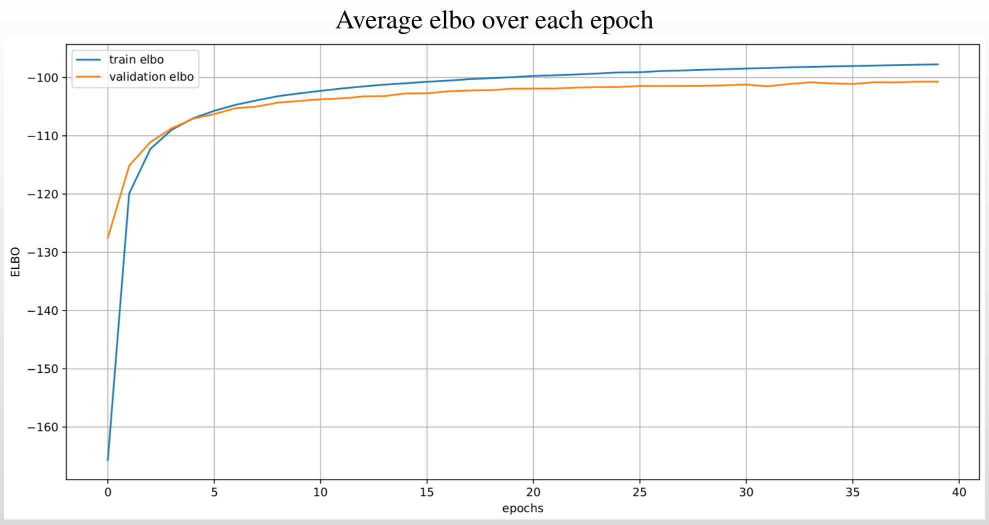
For more implementation details please have a look into `a3_vae_template.py`

Question 1.13

Plot the estimated lower-bounds of your training and validation set as training progresses — using a 20-dimensional latent space.

(Hint: Think about what reasonable average elbo values are for a VAE with random initialized weights (i.e., the predicted mean of the output distribution is random) to make sure you compute the correct lower bound over multiple batches)

Answer 1.13

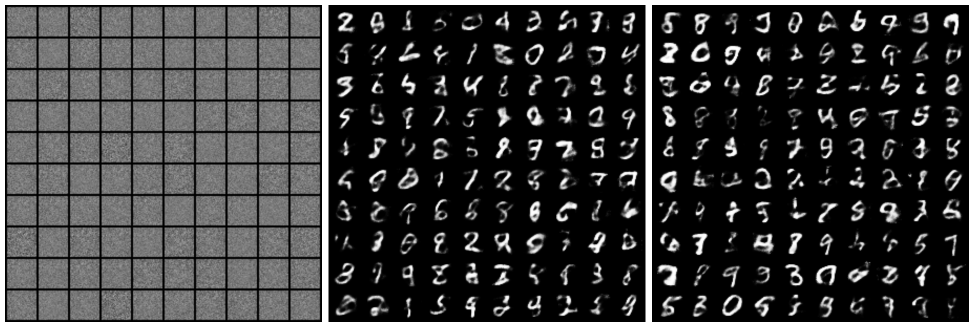


Question 1.14

Plot samples from your model at least three points throughout training (before training, half way through training, and after training). You should observe an improvement in the quality of samples.

Answer 1.14

Observations are shown below in Figure 2:



(a) Start (b) Halfway (c) Termination

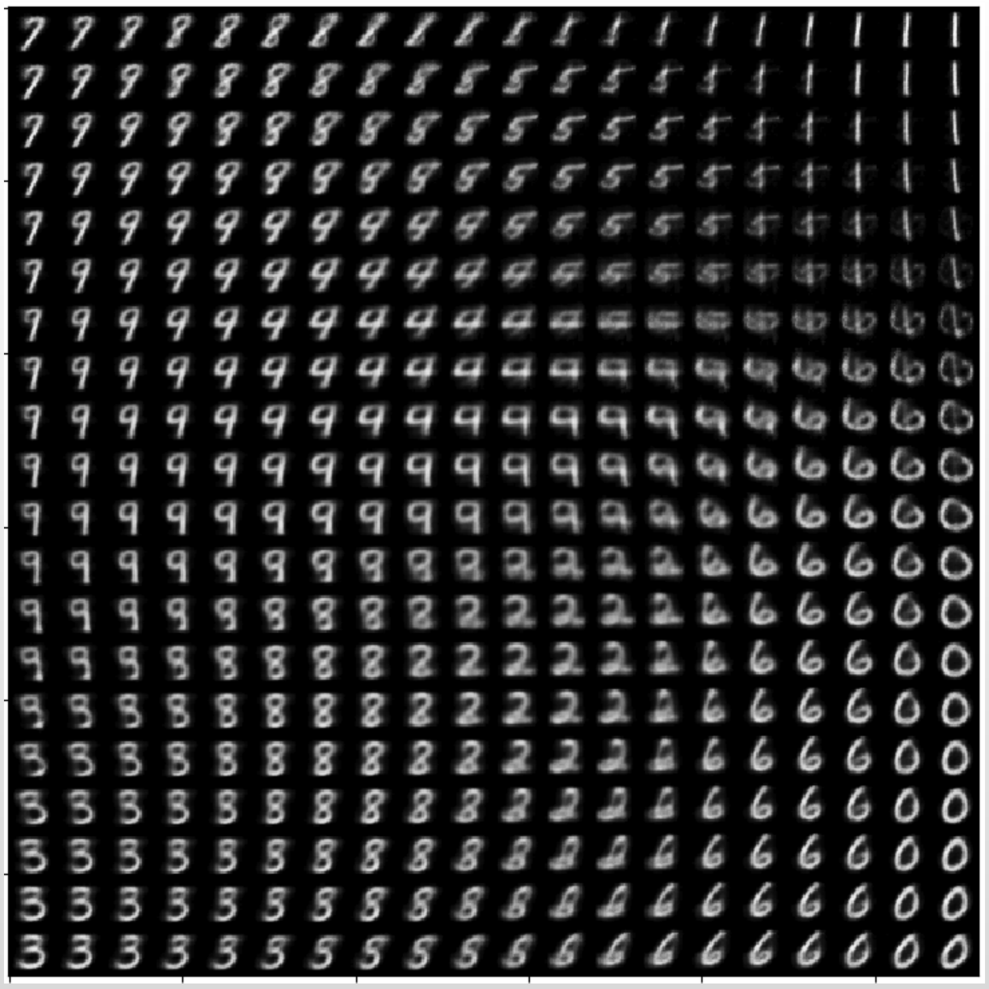
Figure 2: Reconstructed samples during training.

Question 1.15

Train a VAE with a 2-dimensional latent space (or encoding). Use this VAE to plot the data manifold as is done in Figure 4b of [2]. This is achieved by taking a a two dimensional grid of points in Z-space, and plotting $f_\theta(Z) = \mu|Z$. Use the percent point function (ppf, or the inverse CDF) to cover the part of Z-space that has significant density.

Answer 1.15

.



2 Generative Adversarial Networks

Question 2.1

We can think of the generator and discriminator as functions. Explain the input and output for both. *You do not have to take batched inputs into account. Keep your answer succinct.*

For example: for a normal image classification network the input would be an image and the output a vector ρ where ρ_i indicates the probability of the i_{th} class for the given image.

Answer 2.1

Let's define Generator as $G(\mathbf{z}, \theta_1)$ and Discriminator as $D(\mathbf{x}, \theta_2)$ i.e both can be considered as functions with different parameters θ_1, θ_2. Generator maps $G : \mathbf{z} \rightarrow \tilde{\mathbf{x}}$ i.e it's a mapping from one vector space to another vector space. Dimension of $\tilde{\mathbf{x}}$ could vary depending upon the problem statement, in our case it is MNIST digits. Conversely, Discriminator functions $D : \mathbf{x} \rightarrow s$ i.e it maps a vector to a scalar s in the range (0,1). In simple terms it is a binary classifier, classifying fake (0) and true (1).

2.1 Training objective: A Minimax Game

Question 2.2

Explain the two terms in the GAN training objective defined in Equation 15.

Answer 2.2

Generator and Discriminator these two models compete against each other during the training process: the generator $\mathcal{G}$ is trying hard to trick the discriminator, while the critic model $\mathcal{D}$ is trying hard not to be cheated. This interesting zero-sum game between two models motivates both to improve their functionalities.

On one hand, we want to make sure the discriminator $\mathcal{D}$ decisions over real data are accurate by maximizing $\mathbf{E}_{x \sim p_{data}(x)}[\log D(x)]$. Meanwhile, given a fake sample $G(z)$,$z \sim p_z(z)$, the discriminator is expected to output a probability, $D(G(z))$, close to zero by maximizing $\mathbf{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.

On the other hand, the generator is trained to increase the chances of D producing a high probability for a fake example, thus to minimize $\mathbf{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.
When combining both aspects together, D and G are playing a minimax game, we obtain eq 15.

Question 2.3

What is the value of $V(D, G)$ after training has converged.

Answer 2.3

$-2 \log 2$

Question 2.4

Early on during training, the $\log(1 - D(G(Z)))$ term can be problematic for training the GAN. Explain why this is the case and how it can be solved.

Answer 2.4

When the discriminator is perfect, we are guaranteed with $D(x) = 1, \forall x \sim p_{data}$ and $D(G(z)) = 0, \forall z \sim p_z$. Therefore the loss function falls to zero and we end up with no gradient to update the loss during learning iterations as shown in [1]. As a result, training a GAN faces a dilemma:

(a) If the discriminator behaves badly, the generator does not have accurate feedback and the loss function cannot represent the reality.

(b) If the discriminator does a great job, the gradient of the loss function drops down to close to zero and the learning becomes super slow or even jammed i.e D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(z)))$ saturates.

Solution to (b) is rather than training G to minimize $\log(1 - D(G(z)))$ we can train G to maximize $\log D(G(z))$. For (a) we need to make sure we are properly setting up for training environment and other hyperparameters etc for proper training of the GANs.

2.2 Building a GAN

Question 2.5

Build a GAN in PyTorch, and train it on the MNIST data. Start with the template in a3_gan_template.py in the code directory for this assignment. Provide a short (no more than ten lines) description of your implementation.

You will get full points on this question if your answers to the following questions indicate that you successfully trained the VAE and your description is sufficient, however, do not forget to include your code in your final submission.

Answer 2.5

(a) A neural network $G(\mathbf{z}, \theta_1)$ is used to model the generator under `Generator` class. Where $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a Gaussian noise. Generator maps z to the desired data space $\tilde{\mathbf{x}}$.

(b) Conversely, a second neural network $D(\mathbf{x}, \theta_2)$ models the discriminator (`Discriminator` class) and outputs the probability that the data came from the real dataset, in the range (0,1).

(c) Generators weight's are optimized to maximize the probability that any fake image is classified as belonging to the real dataset. Formally this means that the loss/error function used for this network maximizes $\log(D(G(\mathbf{z})))$. Which is being taken care by `nn.BCELoss`. one has to make sure while doing bakward pass, only update the weights for Generator only by `loss_G.backward()`.

(d) Furthermore discriminators weights are updated as to maximize the probability that any real data input $\mathbf{x}$ is classified as belonging to the real dataset, while minimizing the probability that any fake image is classified as belonging to the real dataset. In more technical terms, the loss/error function used maximizes the function $\log(D(\mathbf{x}))$, and it also minimizes $\log(1 - D(G(\mathbf{z})))$. In backward pass only update `loss_D.backward()` and also make sure to detach your output from generator in order avoid updates for generator in this step.

(e) Complete implementation in `a3_gan_template.py`

Question 2.6

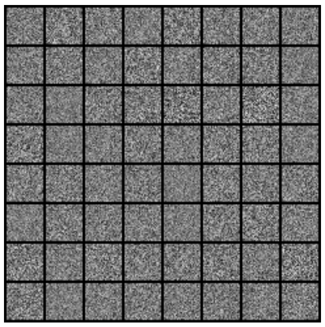
Sample 25 images from your trained GAN and include these in your report (see Figure 2a in [1] for an example. Do this at the start of training, halfway through training and after training has terminated.

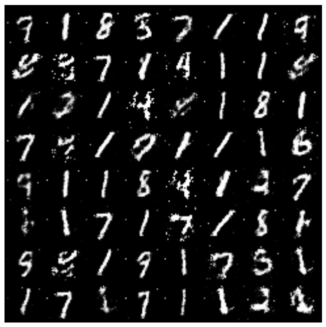
Answer 2.6

Samples shown below in Figure 3.

Question 2.7

Sample 2 images from your GAN (make sure that they are of different classes). Interpolate between these two digits in latent space and include the results in your report. Use 7 interpolation steps, resulting in 9 images (including start and end point).


(a) Start


(b) Halfway

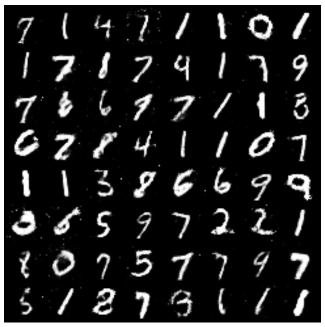
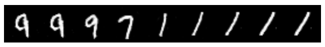
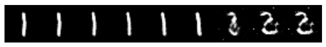

(c) Termination

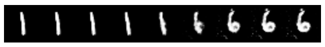
Figure 3: Reconstructed samples during training in GAN.

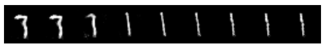
Answer 2.7

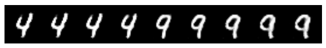
Interpolation below as shown in Figure 4:


(a) Interpolation 1


(b) Interpolation 2


(c) Interpolation 3


(d) Interpolation 4


(e) Interpolation 5

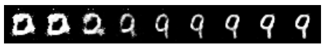

(f) Interpolation 6

Figure 4: Interpolation between 2 randomly generated digits.

3 Conclusion

Question 3.1

Write a short conclusion to your report in which you compare VAEs and GANs. Try to use no more than 10 lines for your conclusion.

Answer 3.1

(a) Qualitatively if we compare Figure 2 and Figure 3 we could easily observe the difference between results. Reconstruction from VAE's seems to be quite blurry looks more of average. On the other hand GANs have very nice reconstruction but are prone to noises as observed. **Below points are more from a theoretical perspective.**

(b) GANs require differentiation through the visible units, and thus cannot model discrete data, while VAEs require differentiation through the hidden units, and thus cannot have discrete latent variables.

(c) Like generative adversarial networks, variational autoencoders pair a differentiable generator network with a second neural network. Unlike generative adversarial networks, the second network in a VAE is a recognition model that performs approximate inference.

(d) An advantage for VAEs is that there is a clear and recognized way to evaluate the quality of the model (log-likelihood, either estimated by importance sampling or lower-bounded). Right now it's not clear how to compare two GANs (Generative Adversarial Networks) or compare a GAN and other generative models except by visualizing samples.

References

[1] Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *CoRR*, abs/1701.04862, 2017.

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