Deep Learning assignment 3

Vasileios Charatsidis
charatsidisvasileios@gmail.com
12148288

Question 1.1

1

- Autoencoders learn a "compressed representation" of input (could be image, text sequence etc.) automatically by first compressing the input (encoder) and decompressing it back (decoder) to match the original input. The learning is aided by using distance function that quantifies the information loss that occurs from the lossy compression. So learning in an autoencoder is a form of unsupervised learning (or self-supervised as some refer to it) - there is no labeled data.
- Instead of just learning a function representing the data (a compressed representation) like autoencoders, variational autoencoders learn the parameters of a probability distribution representing the data. Since it learns to model the data, we can sample from the distribution and generate new input data samples. So it is a generative model like, for instance, GANs.

2

As explained above the simple autoencoder just compress the data in a latent space and then decompress it back to data it came from, it does not generate new data. Auto encoder just does non linear PCA.

3

Of course someone can use VAEs to learn latent representations. VAEs are known to give representations with disentangled factors. This happens due to isotropic Gaussian priors on the latent variables. Modeling them as Gaussians allows each dimension in the representation to push themselves as farther as possible from the other factors.

4

The main difference is that VAEs learn a probability distribution that can represent the data, it cannot be more general than that. Simple AEs just learn to compress a single data to its compressed representation.

1.2

ancestral (or forward) sampling for sampling: Given a probability $p(x_1, ..., x_n)$ specified by a Bayes net, we sample variables in topological order. We start by sampling the variables with no parents; then we sample from the next generation by conditioning these variables' CPDs to values sampled at the first step. We proceed like this until all variables have been sampled. Importantly, in a Bayesian network over n variables, forward sampling allows us to sample from the joint distribution in linear time by taking exactly 1 multinomial sample from each CPD.

So in our case we will first sample from p(z) and then from p(x|z).

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

1.3

In the variational autoencoder, p is specified as a standard Normal distribution with mean zero and variance one, or p(z) = Normal(0,1). If the encoder outputs representations z that are different than those from a standard normal distribution, it will receive a penalty in the loss. This regularizer term means 'keep the representations z of each digit sufficiently diverse.

f is deterministic but if z is random and θ is fixed, then $f(z;\theta)$ is a random variable in the space X, hence the non parametrized p(Z) is not a restrictive assumption. $f(z;\theta)$ will be like the X's in our dataset.

In general, and particularly early in training, our model will not produce outputs that are identical to any particular X. By having a Gaussian distribution, we can use gradient descent (or any other optimization technique) to increase P(X) by making f(z;) approach X.

1.4 a

Evaluating the quality of deep generative models is a hard thing to do usually because you don't know the actual data distribution. Instead, you just have a bunch of samples from it. One way to evaluate models is to look at the marginal likelihood of your model. That is, if your model is probabilistic conditional on some known random variable, we can estimate the probability of a data point X occurring given the model by:

- Sampling many Z from the known latent distribution, and
- Computing the average value of p(X|Z) for all the sampled Z.

where X is our resultant sample from our data distribution (i.e. training data sample), Z is something we know how to sample from e.g. a Gaussian. More precisely, you can estimate the marginal likelihood via Monte Carlo sampling for a single data point X like so:

$$logp(x_n) = \int p\left(x_n|z_n\right) p\left(z_n\right) dz_n \approx \tfrac{1}{N} \sum_{i=1}^{N} p\left(x|z_i\right), \quad z_i \sim \mathcal{N}(0,1)$$

1.4 b

The curse of dimensionality! Our standard VAE has a vector of latent Z variable distributed as independent standard Gaussians. To get good coverage of the latent space, we would have to sample an exponential number of samples from the generator. Obviously not something we want to do for a 100 dimension latent space.

1.5 a

when μ_q is very close to 0 and σ_q^2 very close to 1 then the $D_{KL}(q||p)$ will be small.

1.5 b

For the univariate case :

$$KL(f||g) = \int f(x) \log_e \tfrac{f(x)}{g(x)}\, dx = \int f(x) \log_e f(x)\, dx - \int f(x) \log_e g(x)\, dx$$

Part 1:

$$\int f(x) \log_e f(x)\, dx = \int f(x) \log_e \left(\tfrac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) dx =$$
$$\int f(x)\Big(\big(\log_e \left(\tfrac{1}{\sqrt{2\pi\sigma^2}}\right)\big) + \big(-\tfrac{(x-\mu)^2}{2\sigma^2}\big)\Big)\, dx =$$
$$\int f(x)\big(-\log_e\left(\sigma\sqrt{2\pi}\right)\big)\, dx + \int f(x)\big(-\tfrac{(x-\mu)^2}{2\sigma^2}\big)\, dx$$

$-\log_e(\sigma\sqrt{2\pi})\int f(x)\,dx + \int f(x)\big(-\frac{(x-\mu)^2}{2\sigma^2}\big)\,dx$

$\int f(x)\,dx = 1$ Since f and g are probability density functions.

$-\log_e(\sigma\sqrt{2\pi}) + \frac{1}{2\sigma^2}\big(-\int f(x)x^2\,dx + 2\int f(x)x\mu\,dx - \int f(x)\mu^2\,dx\big)(1)$

But $\sigma^2 = E[x^2] - (E[x])^2 = \int f(x)x^2\,dx - (\int f(x)x\,dx)^2 \rightarrow$

$\int f(x)x^2\,dx = \sigma^2 + \mu^2$

So (1) becomes: $-\log_e(\sigma\sqrt{2\pi}) + \frac{2}{2\sigma}\big(-\sigma^2 - \mu^2 + 2\mu^2 - \mu^2) = -\frac{1}{2}\big(\log_e(2\pi\sigma^2) + 1\big)$

Part 2:

$-\int f(x)\log_e g(x)\,dx$ = $\int f(x)\log_e(\frac{1}{\sqrt{2\pi\tau^2}}e^{-\frac{(x-\nu)^2}{2\tau^2}})\,dx$

$\frac{1}{2}\log_e 2\pi\tau^2 - \int f(x)\big(-\frac{(x-\nu)^2}{2\tau^2}\big)\,dx$

$\frac{1}{2}\log_e 2\pi\tau^2 - \frac{1}{2\tau^2}\Big(-\int f(x)x^2\,dx + 2\int f(x)x\nu\,dx - \int f(x)\nu^2\,dx\Big)(2)$

Similarly with part one we can replace all integrals using mean and variance since f and g are pdfs.

So (2) becomes:

$\frac{1}{2}\log_e 2\pi\tau^2 + \frac{1}{2\tau^2}(\sigma^2 + \mu^2 - 2\mu\nu + \nu^2) =$

$\frac{1}{2}\log_e 2\pi\tau^2 + \frac{1}{2\tau^2}(\sigma^2 + (\mu-\nu)^2)$

Combining all together:

$KL = -\frac{1}{2}\big(\log_e 2\pi\sigma^2 + 1\big) + \frac{1}{2}\log_e 2\pi\tau^2 + \frac{\sigma^2+(\mu-\nu)^2}{2\tau^2} \rightarrow$

$KL = \frac{1}{2}log_e(\frac{\tau}{\sigma})^2 + \frac{\sigma^2+(\mu-\nu)^2}{2\tau^2} - \frac{1}{2} \rightarrow$

$KL = \log_e\frac{\tau}{\sigma} + \frac{\sigma^2+(\mu-\nu)^2}{2\tau^2} - \frac{1}{2}$

For the multivariate case:

$$KL = \int\left[\frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\right]\times p(x)dx$$

$$= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}\operatorname{tr}\left\{E\left[(x-\mu_1)(x-\mu_1)^T\right]\Sigma_1^{-1}\right\} + \frac{1}{2}E\left[(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\right]$$

$$= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}\operatorname{tr}\{I_d\} + \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2) + \frac{1}{2}\operatorname{tr}\left\{\Sigma_2^{-1}\Sigma_1\right\}$$

$$= \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \operatorname{tr}\left\{\Sigma_2^{-1}\Sigma_1\right\} + (\mu_2-\mu_1)^T\Sigma_2^{-1}(\mu_2-\mu_1)\right]$$

for

$$q = \mathcal{N}(\mu_q, \Sigma_q)$$

and

$$p = \mathcal{N}(0,1)$$

$$D_{KL}\left(\mathcal{N}\left(\mu_q, \Sigma_q\right) \| \mathcal{N}\left(0, I_D\right)\right) = \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_q\right) + \left(\mu_q\right)^T\left(\mu_q\right) - D - \log\det\left(\Sigma_q\right)\right)$$

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

It is called lower bound because

$$D_{\mathrm{KL}}\left(q\left(Z|x_n\right) \| p\left(Z|x_n\right)\right) \geq 0$$

so that means

$$\log p(x_n) \geq \mathbb{E}_{q(z|x_n)}\left[\log p\left(x_n|Z\right)\right] - D_{\mathrm{KL}}\left(q\left(Z|x_n\right) \| p(Z)\right)$$

always. So the right hand side is the Lower bound of $\log p(x_n)$.

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

To compute and minimize the Kullback-Leibler divergence between the approximate and exact posteriors is computationally intractable. Instead, we can maximize the ELBO which is equivalent (but computationally tractable).

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?

Either the log of probability will go up so we generate distribution that resemble the data well or $D_K L$ goes down so

$$q\left(Z|x_n\right)$$

diverges less from

$$p\left(Z|x_n\right)$$

.

1.9 Explain why the names reconstruction and regularization are appropriate for these two losses

$$\mathcal{L}_n^{\mathrm{reg}} = D_{\mathrm{KL}}\left(q_\phi\left(Z|x_n\right) \| p_\theta(Z)\right)$$

Is called regularization because it bounds the distribution to meaningful values that are close each to each other. Without it the z might be meaningless. In other words it encourages our learned distribution q(z|x) to be similar to the true prior distribution p(z).

The

$$\mathcal{L}_n^{\text{recon}} = -\mathbb{E}_{q_\phi(z|x_n)}\left[\log p_\theta\left(x_n|Z\right)\right]$$

Is called reconstruction loss because it penalizes reconstruction error or in other words maximizes the reconstruction likelihood, since it enforces q_ϕ to be close to p_θ.

1.10 Write down expressions (including steps) for L_n^{recon} and L_n^{reg} such that we can minimize our final objective. Make any approximation explicit.

Using one sample: $p_\theta(c_n|Z) = \prod_{m=1}^{M} f_\theta(z)_m^{x_n^{(m)}}\left(1 - f_\theta(z)_m\right)^{\left(1-x_n^{(m)}\right)}$

so the $\mathcal{L}_n^{\text{recon}} = -\log p_\theta(c_n|Z) = -\sum_{m=1}^{M}\left(x_n^{(m)}\log f_\theta(z)_m + \left(1 - x_n^{(m)}\right)\log\left(1 - f_\theta(z)_m\right)\right)$

and $\mathcal{L}_n^{\text{reg}} = \frac{1}{2}\left(\text{tr}\left(\Sigma\left(x_n\right)\right) + \left(\mu\left(x_n\right)\right)^T\left(\mu\left(x_n\right)\right) - D - \log\det\left(\Sigma\left(x_n\right)\right)\right)$

$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n^{\text{recon}} + \mathcal{L}_n^{\text{reg}}$

$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\left(\text{tr}\left(\Sigma\left(x_n\right)\right) + \left(\mu\left(x_n\right)\right)^T\left(\mu\left(x_n\right)\right) - D - \log\det\left(\Sigma\left(x_n\right)\right)\right) -$

$\sum_{m=1}^{M}\left(x_n^{(m)}\log f_\theta(z)_m + \left(1 - x_n^{(m)}\right)\log\left(1 - f_\theta(z)_m\right)\right)$

1.11 a $\nabla_\phi\mathcal{L}$

We need $\nabla_\phi\mathcal{L}$ to to adjust the weights of the encoder through back prop.

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

We need to back-propagate the error through a layer that samples z from Q(z|X), which is a non-continuous operation and has no gradient. Stochastic gradient descent via back propagation can handle stochastic inputs, but not stochastic units within the network.

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

The problem in evaluating this quantity is the fact that the expectation is taken wrt a distribution with parameters θ and we can't compute the derivative of that stochastic quantity.

Reparametrization trick solves the problem by moving the sampling to an input layer. Reparameterization gradients also known as pathwise gradients allow us to compute this by re-writing the samples of the distribution p_θ in terms of a noise variable ϵ, that's independent of θ.

$$\begin{aligned}
\varepsilon &\sim q(\varepsilon) \\
x &= g_\theta(\varepsilon) \\
\nabla_\theta\mathbb{E}_{x\sim p_0(x)}[f(x)] &= \nabla_\theta\mathbb{E}_{x\sim q(\varepsilon)}\left[f\left(g_\theta(\varepsilon)\right)\right] \\
&= \mathbb{E}_{x\sim q(\varepsilon)}\left[\nabla_\theta f\left(g_\theta(\varepsilon)\right)\right]
\end{aligned}$$

Thus, x is reparameterized as a function of ϵ and the stochasticity of p_θ is pushed to the distribution $q(\epsilon)$ where q can be chosen as any random noise distribution, eg a standard Gaussian $\mathcal{N}(0, 1)$.

1.12 Provide a short description of your implementation.

The architecture is inspired by the paper Auto-Encoding Variational Bayes Kingma and Welling 2014.

- Encoder: Consists of a linear layer with input dim the dimension of our input data 28 * 28 = 784 and output dimension 500. Then the results is passed through a ReLU layer and then there are 2 linear layers. One for the mean and one for the standard deviation that have as output a z dimensional vector.
- Reparametrization trick: The output of the encoder is the mean and the variance we perform the reparametrization trick so as $z = std * e + mean$ where e comes from a normal distribution with mean = 0 and variacne = 1.
- Decoder: receives z into a linear layer with output 500. The results is passed through a tanh layer and then another linear layer that outputs 784 dim vector wich match the dimension of the mnist. Finally this is passed through a sigmoid unit and returned.

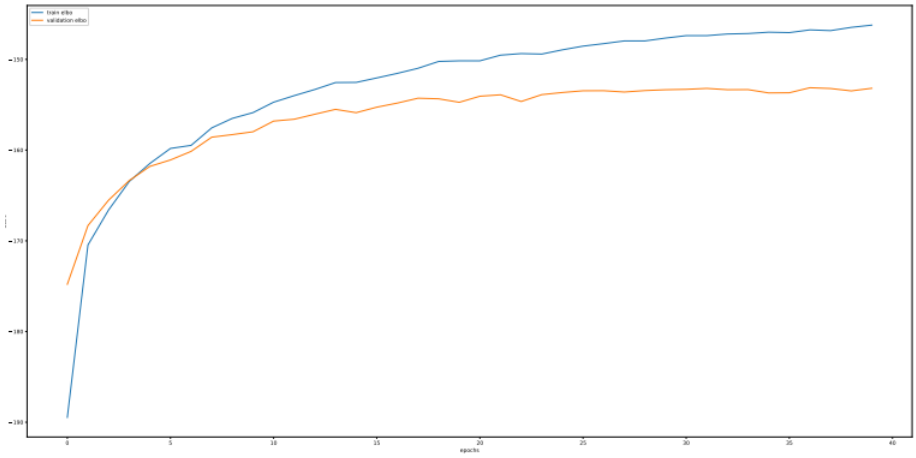


Figure 1: Elbo

(a) Binary, epoch 0 (b) Binary, epoch 20 (c) Binary, epoch 39

Figure 2: Plots of binary samples, during and at the end (c) of training in a 20-dimension space

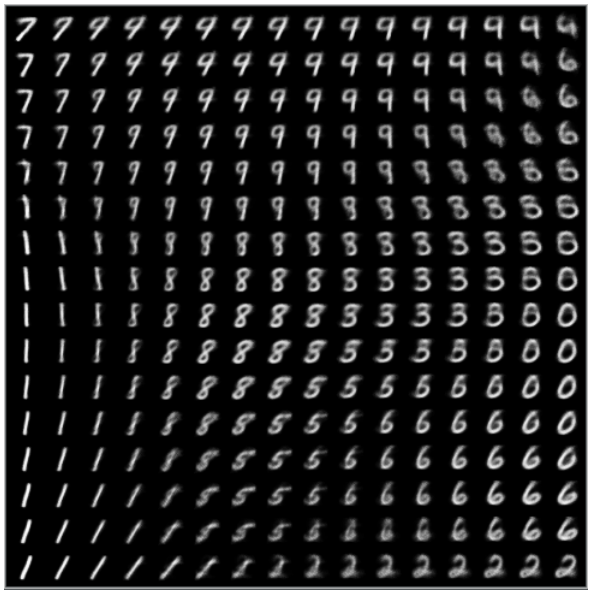


Figure 3: Manifold

1.13 plot

1.14 results through start mid and end of the training process

1.15 Manifold

2 Generative Adversarial Networks

2.1 Explain the input and output of the Generator and Discriminator

1. The Generator takes in a z-dimensional array of randomly sampled points from a uniform distribution and the output is what we want to achieve. Most likely an image since everybody nowdays use GANs to generate images.

2. This generated image is fed into the Discriminator alongside a stream of images taken from the dataset.
3. The Discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

2.2 Explain the two terms in the GAN training objective defined

$$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{p_{\text{data}}(x)}[\log D(X)] + \mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$$

- The first term $\mathbb{E}_{p_{\text{data}}(x)}[\log D(X)]$ is the expectation over data of $\log D(X)$ where D(X) is the Discriminator output for real data x. In other words the expected chance that the discriminator will classify a real data as real.
- The second term is the expectation of z drawn from p(z), so samples from our generator network and D(G(Z)) is the discriminator output for generated fake data G(Z). The expected chance that the discriminator will classify a data generated from the Generator as fake.

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

Its the Nash Equilibrium where the discriminator cannot discriminate real images from fake images, generated from the generator, anymore.

That means that D(X) = D(G(Z)) = 1/2. That means that also 1 - D(G(Z)) = 1/2. Then the loss become:

$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{p_{\text{data}}(x)}[\log(1/2)] + \mathbb{E}_{p_z(z)}[\log(1/2)] = \log\left(\frac{1}{2}\right) + \log\left(\frac{1}{2}\right) = 2\log\left(\frac{1}{2}\right) = -\log(4) = -1.3863$

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.

In the start the samples of the generator are pretty random, when a sample is likely fake the D(G(Z)) is almost 1 and the $log(1 - D(G(Z)))$ becomes almost 0. The graph of log close to 0 has a very small gradient. This means that the generator will learn slowly or not at all.

The way to solve this is to maximize the chance that the descriminator fails recognise a generated image as fake. The graph of log(x) is a lot steeper when x is close to 0. That means that we are going to have bigger gradients and learn faster.

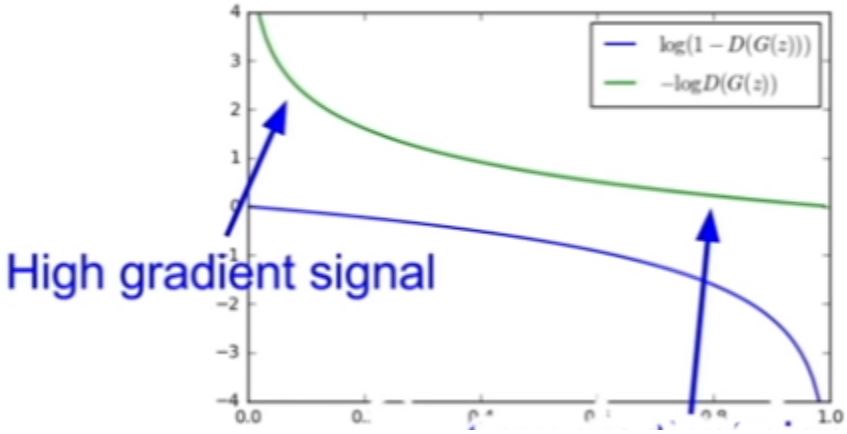


Figure 4: comparison of log(D(G(Z)) and log(1-D(G(Z))

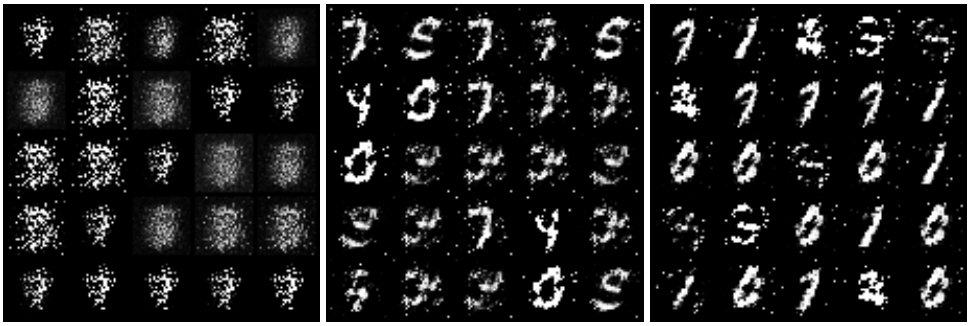
2.5 Build a GAN in PyTorch, and train it on the MNIST data.

The implmentations used was the proposed.

- Discriminator:
  - Linear 784 -> 512
  - LeakyReLU(0.2)
  - Linear 512 -> 256
  - LeakyReLU(0.2)
  - Linear 256 -> 1
  - Sigmoid
- Generator:
  - Linear 100 -> 128
  - LeakyReLU(0.2)
  - Linear 128 -> 256
  - Bnorm
  - LeakyReLU(0.2)
  - Linear 256 -> 512
  - Bnorm
  - LeakyReLU(0.2)
  - Linear 512 -> 1024
  - Bnorm
  - LeakyReLU(0.2)
  - Linear 1024 -> 768
  - Tanh

The only trick used is one-sided label smoothing in the loss of Discriminator. We multiply the targets of the discriminator with 0.9 just to slow down its training so that it trains hand to hand with the generator. Without this trick the learning stops because the generator does not receive any signal since the discriminator recognize all generated-fake images.

2.6



(a) After batch 2000 (b) After batch 102000 (c) After batch 186000

Figure 5: Samples from GAN at begging, middle and end of training

3 Generative Normalizing Flows

3.1 Rewrite the equations

$z = f(x)$

$p(x) = p\left(f^{-1}(z)\right) \cdot \left|\det J\left(f^{-1}(z)\right)\right|$

$\log p(z) = \log p\left(f^{-1}(z)\right) + \log \left|\det \left(J\left(f^{-1}(z)\right)\right)\right|$

3.2 What are the constraints that have to be set on the function f to make this equation computable?

1. x and z need to be continuous and have the same dimension.
2. The transformation must be invertible.
3. Computing the determinant of the Jacobian needs to be efficient (and differentiable).

3.3 Even if we ensure property mentioned in the previous question, what might be a computational issue that arises when you optimize your network using the objective you derived in question 3.1

The jacobian might be really costly to compute.

3.4 The change-of-variables formula assumes continuous random variables, however, images are often stored as discrete integers. What might be the consequence of that and how would you fix it?

A solution to this problem is to first convert the discrete data distribution into a continuous distribution via a process called "dequantization," and then model the resulting continuous distribution using the continuous density model (Uria et al., 2013; Dinh et al., 2016; Salimans et al., 2017).

Dequantization is usually performed in prior work by adding uniform noise to the discrete data over the width of each discrete bin: if each of the D components of the discrete data x takes on values in 0, 1, 2, . . . , 255, then the dequantized data is given by y = x+u, where u is drawn uniformly from [0, 1)D. Theis et al. (2015)

3.5 Not Done

3.6 Not Done

3.7 Not Done

3.8 Not Done

4 Conclusion

GANs, VAEs and GNFs are generative models, which means they learn a given data distribution rather than its density.

VAEs learn a given distribution comparing its input to its output, this is good for learning hidden representations of data, but is pretty bad for generating new data. Mainly because we learn an averaged representation of the data thus the output becomes pretty blurry.

GANs take an entirely different approach. They use another network (so-called Discriminator) to measure the distance between the generated and the real data. Basically what it does is distinguishing the real data from the generated. The generators goal then is learning to convince the Discriminator into believing it is generating real data. GAN's main problem is that it is difficult to train. Difficult to synchronize the adversarial game so that the Discriminator and Generator get both better at the propper pace.

VAEs are more suitable for compressing data to lower dimensions or generating semantic vectors from it. Where GANs are more suitable for generating data.

GNFs is a type of method that combines the best of both worlds, allowing both feature learning and tractable marginal likelihood estimation.

In normalizing flows, we wish to map simple distributions (easy to sample and evaluate densities) to complex ones (learned via data). The key idea in the design of a flow model is to form f by stacking individual simple invertible transformations.The change of variables formula describe how to evaluate densities of a random variable that is a deterministic transformation from another variable.

References

[1] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. International Conference on Learning Representations, ICLR, $2017.1, 11$

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. International Conference on Learning Representations $(ICLR), 2014.1, 6, 7$