IFT 6135 - Homework 3

Antoine Chehire, Philippe Marchandise, Adel Nabli

22/04/2019

Link of the Github where the code used is stored: `https://github.com/achehire/Deep_1_AC_AN_PW`

1 Problem 1

1. In order to estimate the Jensen Shannon Divergence, we will train a parametrized discriminator D_θ trained to maximize the objective function (1). After training, provided that our discriminator has sufficient capacity, the objective function will approximate the JSD between p and q.

   - **First**, we implement the objective function
   - **Second**, we create a discriminator D_θ in the form of an MLP with 3 hidden layers. As we deduce from (1) that $\forall x, D_\theta(x) \in]0,1[$, the output non-linearity is a sigmoid
   - **Third**, we implement a function that, provided 2 samplers of the distributions p, q, compute the JSD by training the discriminator (for 30 000 steps) and returns the last objective value computed.

   The objective function that our neural network should optimize in the case we are using the Jensen Shannon Divergence is:

   $$obj = \log 2 + \frac{1}{2}\mathbb{E}_{x\sim p}[\log D_\theta(x)] + \frac{1}{2}\mathbb{E}_{y\sim q}[\log(1 - D_\theta(y))] \tag{1}$$

   In the case we are using batches of size m, this expression is approximated by:

   $$obj \simeq \log 2 + \frac{1}{2m}\sum_{i=1}^{m}\log D_\theta(x_i) + \frac{1}{2m}\sum_{i=1}^{m}\log(1 - D_\theta(y_i)) \tag{2}$$

2. We procede in the same way than **Q1.1**, except that now the objective to maximize is:

   $$\mathbf{E}_{x\sim p}[T_\theta(x)] - \mathbf{E}_{y\sim q}[T_\theta(y)] - \lambda\mathbf{E}_{z\sim r}[(||\nabla_z T_\theta(z)||_2 - 1)^2] \tag{3}$$

   with r the distribution over $z = ax + (1-a)y$, where $x \sim p$, $y \sim q$ and $a \sim U[0,1]$. We also have $\lambda \geq 10$.

   - We fix $\lambda = 10$
   - We change the output activation function compared to before *(here we finish with a linear one so that $\forall x,\ T_\theta(x) \in \mathbb{R}$)*

3. We report our approximations in the graph below:

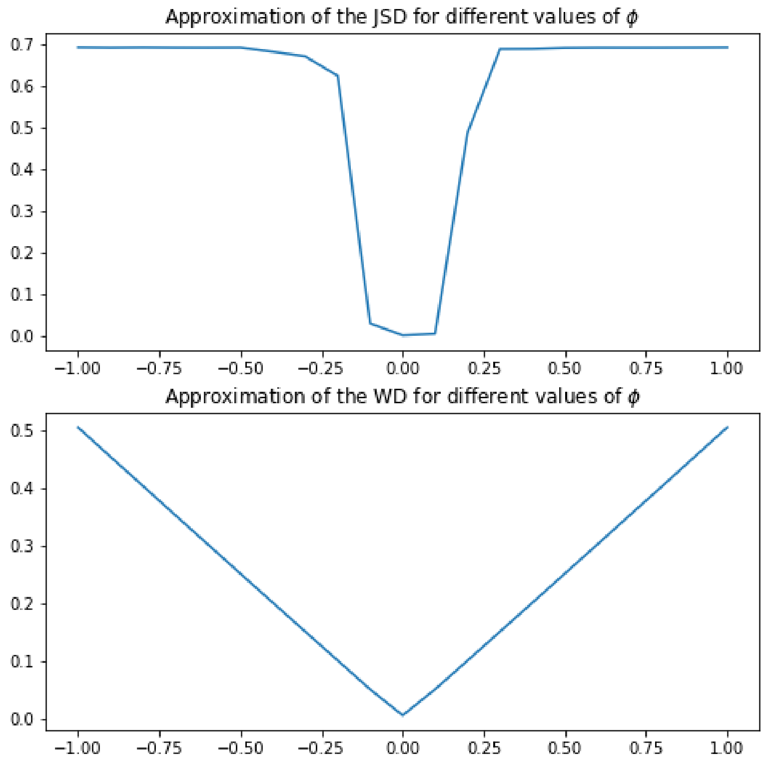

Figure 1: Approximation of the JSD and WD for 21 values of ϕ

4. This time, the discriminator has the following objective function:

$$obj = \mathbb{E}_{x \sim f_1}[\log D_\theta(x)] + \mathbb{E}_{y \sim f_0}[\log(1 - D_\theta(y))] \tag{4}$$

We train it using the two distributions f_0 and f_1

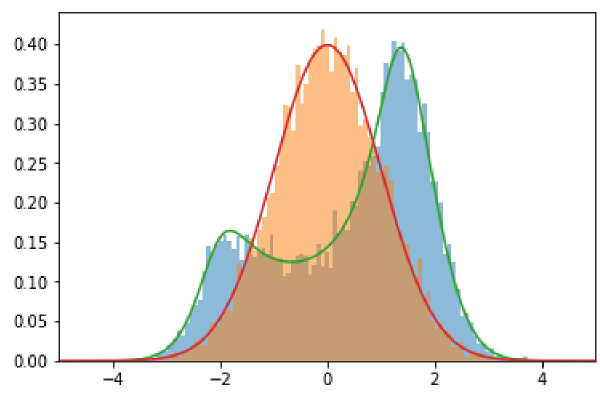

Figure 2: Samples from f_1 and f_0

With the trained discriminator D_θ^*, given f_0, we can approximate f_1 using the following formula:

$$\forall x,\ f_1(x) = f_0(x) \frac{D_\theta^*(x)}{1 - D_\theta^*(x)} \tag{5}$$

which leads to the following results:

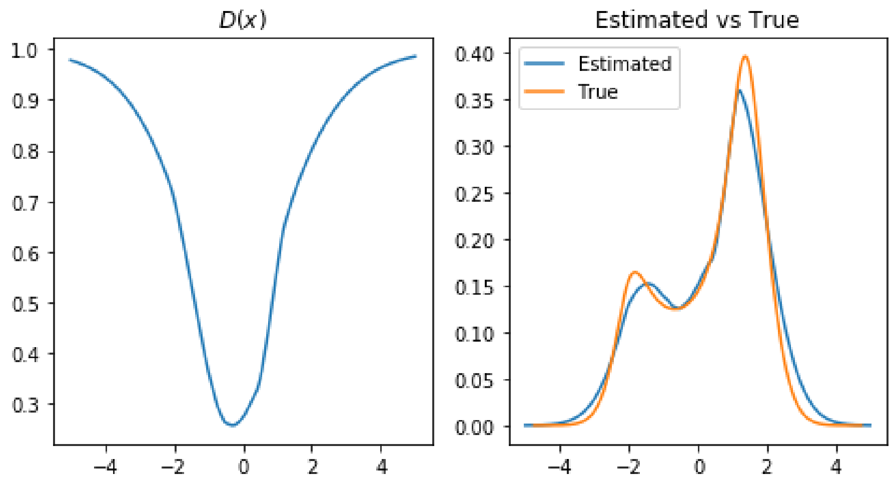

Figure 3: Approximated distribution using the discriminator

2 Problem 2

2.1 Training

We chose to implement the VAE describes in the directions. We trained it using ADAM with a learning rate of 3×10^{-4} to minimize the ELBO loss on 20 epochs.

On the validation set, we achieved an ELBO of: -94.3

2.2 Evaluating log likelihood

Please find the code below:

```python
def estimate_log_px(model, x, z):
    m, d = x.shape
    _, k, l = z.shape

    # First, let's compute p(z):
    pz = norm.pdf(z)  # for each unit
    log_pz = np.sum(np.log(pz), axis=-1)  # Using log-sum trick to get the product

    # Then, let's compute q(z|x) for each unit:
    data = torch.Tensor(x).view(-1, 1, 28, 28)
    encoded_layer = model.encoder(data.cuda())
    mean, log_var = encoded_layer.chunk(2, 1)
    mean, log_var = mean.cpu(), log_var.cpu()
    qz = [norm.pdf(z[i], loc=mean[i], scale=np.exp(log_var[i]/2))
          for i in range(len(mean))]
    qz = np.array(qz)  # for each unit
    log_qz = np.sum(np.log(qz), axis=-1)

    # Then, let's compute p(x|z) for each unit (it's cross entropy):
    samples = torch.Tensor(z).view(-1, 100)
    predict = model.decoder(samples.cuda())
    predict = predict.view(-1, k, d)  # mxkxd
    predict = predict.transpose(0,1)  # kxmxd
    predict = predict.cpu().numpy()
    log_pxz = []
    for pred in predict:
        log_sample = np.sum(x * np.log(expit(pred)) + (1. - x) * np.log(1.0 - expit(pred)),
                            axis=-1)
        log_pxz.append(log_sample)
    log_pxz = np.array(log_pxz)  #kxm
    log_pxz = np.swapaxes(log_pxz, 0, 1)  #mxk

    log_px = log_pxz + log_pz - log_qz  # as we're still using log-sum trick
    # note here that + and - are done elementwise (log_px is an array)
    log_px = np.log(np.mean(np.exp(log_px), axis=1))  # Computing log_px

    return log_px
```

We report below our validation and test ELBO for our model:

- Validation ELBO: -94.3
- Test ELBO: -93.5

And our validation and test log likelihood estimates:

- Validation ELBO: -88.96
- Test ELBO: -88.25

These results make sense as we have our estimated log likelihood greater than the ELBO (which is necessary by definition of ELBO).

3 Problem 3

We encountered a problem when generating samples from our GAN model. Therefore, we can only report results for the VAE model.

We chose to keep the VAE model from the exercise 2. However, instead of binary cross entropy loss, we chose to use the mean squared error loss when training our model.

3.1 Qualitative analysis

First, here is our reference image:

Figure 4: Reference image

Here is the image generated by our VAE:

Figure 5: Generated image

The image is quite blurry compared to the original.

We then analyzed the disentanglement. Here is the image generated by our model:

Figure 6: Generated image

Following the directions, we made small changes on each of the 100 dimensions. In most dimensions, the changes are barely visible. However, in some dimensions, we obtained interesting results showing the disentanglement:

Figure 7: Small change in one dimension

Figure 8: Small change in another dimension

We then report our interpolations. In the latent variable space:

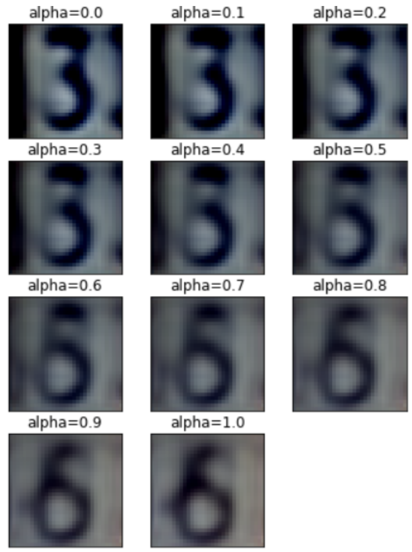

Figure 9: Interpolation in latent space

In the sample space

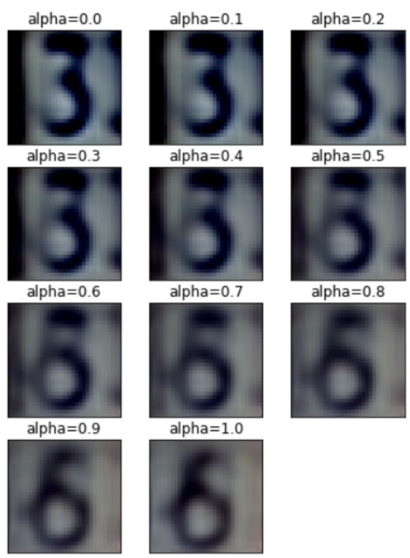

Figure 10: Interpolation in samples space

The changes are actually fairly similar. We notice however that interpolating in the latent space yields darker images in general.