Introduction

In this practical assignment we will study the basics of variational inference and we will use this method for doing approximate inference in a real model. In our case, we will work with a generative model that has the form:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z},$$

where $\mathbf{x}$ is some data, $p_\theta(\mathbf{x} \mid \mathbf{z})$ is a conditional distribution and $p((z))$ is a source of noise.

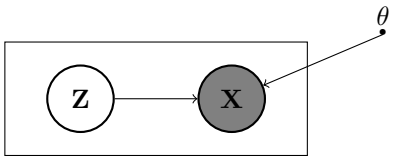

Figure 1: Probabilistic graphical model describing the considered generative model.

Figure 1 depicts the described model. Important considerations are the fact that our source of noise will be a standard L dimensional Gaussian

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I}),$$

and that our data is binary, since it will consists on black and white images: $\mathbf{x} \in \{0, 1\}^D$.
Our approach is to estimate θ given $\mathbf{X}$ using maximum likelihood estimation. Assuming iid data, we know that

$$\hat{\theta} = \arg\max_\theta \log p_\theta(\mathbf{X}) = \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x_i}).$$

Considering that computing $p_\theta(\mathbf{x_i})$ is intractable, we will use the evidence lower bound (ELBO). We know that

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\right] + KL(q_\phi(\mathbf{z} \mid \mathbf{x}) \mid p(\mathbf{z} \mid \mathbf{x})) \\
&\geq E_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\right] \\
&= \mathcal{L}(\mathbf{x}, \theta, \phi),
\end{aligned}$$

where KL stands for the Kullback-Leibler divergence and we have use its positiveness. We will have to add $\mathcal{L}$ for every data sample.

TASKS

2.1 Task 1

In this task we will implement the needed functions to build the autoencoder and comment (if needed) the most remarkable steps.

- `sample_latent_variables_from_posterior`

  Here, we use the reparametrization trick, approximating $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$ using Monte Carlo. To achieve this, we sample from the conditional distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ (which is a Gaussian distribution) and use the expression

  $$z_j^i = \mu_j^\phi(\mathbf{x_i}) + \exp(\log \sigma_j) \cdot \epsilon_i^j,$$

  where $\epsilon_i^j \sim \mathcal{N}(0, 1)$.

- `bernoulli_log_prob`

  In this case, it is important to pre-evaluate the sigmoid function previously to compute the log likelihood of the targets, adding up across the image dimensions.

- `compute_KL`

  The code of this function is pretty straightforward, since we know that we are dealing with Gaussian distributions. The KL divergence between two Gaussians is analitical and can be expressed as:

$$KL(q_\phi(\mathbf{z} \mid \mathbf{x_i}) \mid p(\mathbf{z})) = \frac{1}{2} \sum_{j=1}^{L} \left(\nu_j^\phi \mathbf{x_i} + \mu_j^\phi \mathbf{x_i}^2 - 1 - \log \nu_j^\phi(\mathbf{x_i}) \right)$$

- `vae_lower_bound`

  This is the most complicated function. It performs a forward pass through the data and computes a noisy estimate of the ELBO using a single Monte Carlo sample. We use the previously implemented functions. As indicated in the provided `vae.py` script, we have to follow the next steps:

  1. Compute the encoder output `neural_net_predict` using the data.
  2. Sample latent variables associated to the input data, using the output of the previous step.
  3. Reconstruct the image using `neural_net_predict` again and then computing the log likelihood of the data.
  4. Compute the KL divergence.
  5. Estimate the ELBO (per batch point) by substracting the KL to the data dependent term. We took the decision to multiply this ELBO by N, the batch size, since it is a needed normalization constant. This can be avoided, since optimizing the noisy ELBO or N times the noisy ELBO is the same. It was added for precision.

2.2 Task 2

In this task, we complete the ADAM initialization and write the adam updates in the training loop. We are given a description of the ADAM algorithm in the assignment, so our task is to follow this description. Firstly, we initialize the ADAM parameters:

```
# ADAM parameters
alpha = 1e-3
beta_1 = 0.9
beta_2 = 0.999
epsilon = 1e-8
m = np.zeros_like(flattened_current_params)
v = np.zeros_like(flattened_current_params)
```

Then, we perform the update of the parameters.

```
# ADAM step
m = beta_1 * m + (1 - beta_1) * grad
v = beta_2 * v + (1 - beta_2) * grad**2
hat_m = m / (1 - beta_1**t)
hat_v = v / (1 - beta_2**t)

flattened_current_params += alpha * hat_m / (np.sqrt(hat_v) + epsilon)
```

2.3 Task 3.1

In this task we are asked to generate 25 images from the generative model. To achieve this, we need to draw samples from the prior and then generate $\mathbf{x}$ using the conditional distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$.
In order to get the needed result, we generate 25 samples from the prior $\mathcal{N}(0, 1)$ and then compute the higher dimensional representation using the output of the neural network with the `gen_parameters`

```
# Apply autoencoder with noise images
sampled_z = npr.randn(25, latent_dim)
sampled_x = neural_net_predict(gen_params, sampled_z)
```

```
created_images = sigmoid(sampled_x)

# Save images
save_images(created_images, "output/3_1.png")
```

This provides the output shown in Figure 2.

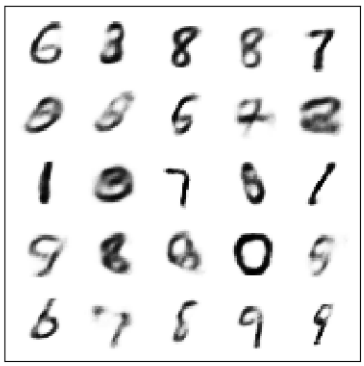

Figure 2: 25 generated samples using the prior distribution $\mathcal{N}(0, I)$.

2.4 Task 3.2

In this task, we are asked to generate 10 image reconstructions using the recognition model and then the generative model. We use the first 10 images from the test set. To achieve this, we have to do part of the process that we did in the noisy ELBO computation. In particular, the first three steps:

```
encoder_output = neural_net_predict(rec_params, test_images[:10])
sampled_z = sample_latent_variables_from_posterior(encoder_output)
sampled_x = neural_net_predict(gen_params,sampled_z)
reconstructed_images = sigmoid(sampled_x)

# Concatenate images and save them
concatenated = np.append(
    test_images[:10],
    reconstructed_images,
    axis=0
)

save_images(concatenated, "output/3_2.png")
```

The output is shown in Figure 3.

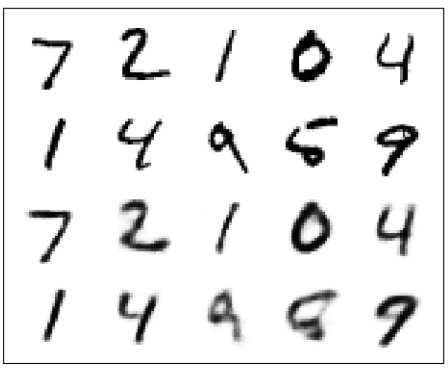

Figure 3: Original data in the first two rows versus reconstruction in the last two rows.

2.5 Task 3.3

In this last task, we have to generate 5 interpolations in the latent space from one image to another. We consider consecutive pairs of images from the test set. The interpolations are obtained by finding the latent representation, considering only the mean of the predictive model $q(\mathbf{z} \mid \mathbf{x})$. Then, we interpolate between the latent representations of the images, $\mathbf{z}_1$ and $\mathbf{z}_2$. We use the *segment* interpolation, which we express in our implementation as:

$$\mathbf{z}^s_{\text{mix}} = (1-s)\mathbf{z}_1 + s\mathbf{z_2}, \quad s \in [0,1].$$

Remark that in $s = 0$, $\mathbf{z}^s_{\text{mix}} = \mathbf{z}_1$ and in $s = 1$ $\mathbf{z}^s_{\text{mix}} = \mathbf{z_2}$. Using this $\mathbf{z}^s_{\text{mix}}$ we can generate the corresponding image using $p_\theta(\mathbf{x} \mid \mathbf{z}^s_{\text{mix}})$.

The following code implements the interpolations:

```
# Obtain the latent representations of each image
encoder_output_1 = neural_net_predict(rec_params, img1)
encoder_output_2 = neural_net_predict(rec_params, img2)

# Obtain the means of each representation
D = np.shape(encoder_output_1)[-1] // 2
mean1 = encoder_output_1[:D]
mean2 = encoder_output_2[:D]

# Interpolate the means
z_interpolations = np.array([
    (1 - s) * mean1 + s * mean2
    for s in np.linspace(0.0, 1.0, num=n_interpolation_steps)
])

# Obtain the interpolated images from the interpolated Z
interpolated_imgs = np.array([
    neural_net_predict(gen_params, z)
    for z in z_interpolations
])
```

It has to be remarked that, after the process of predicting with the neural network, a **sigmoid** has to be applied in order to have in each pixel the probability of being activated (or the intensity). The results are shown in Figure 4.

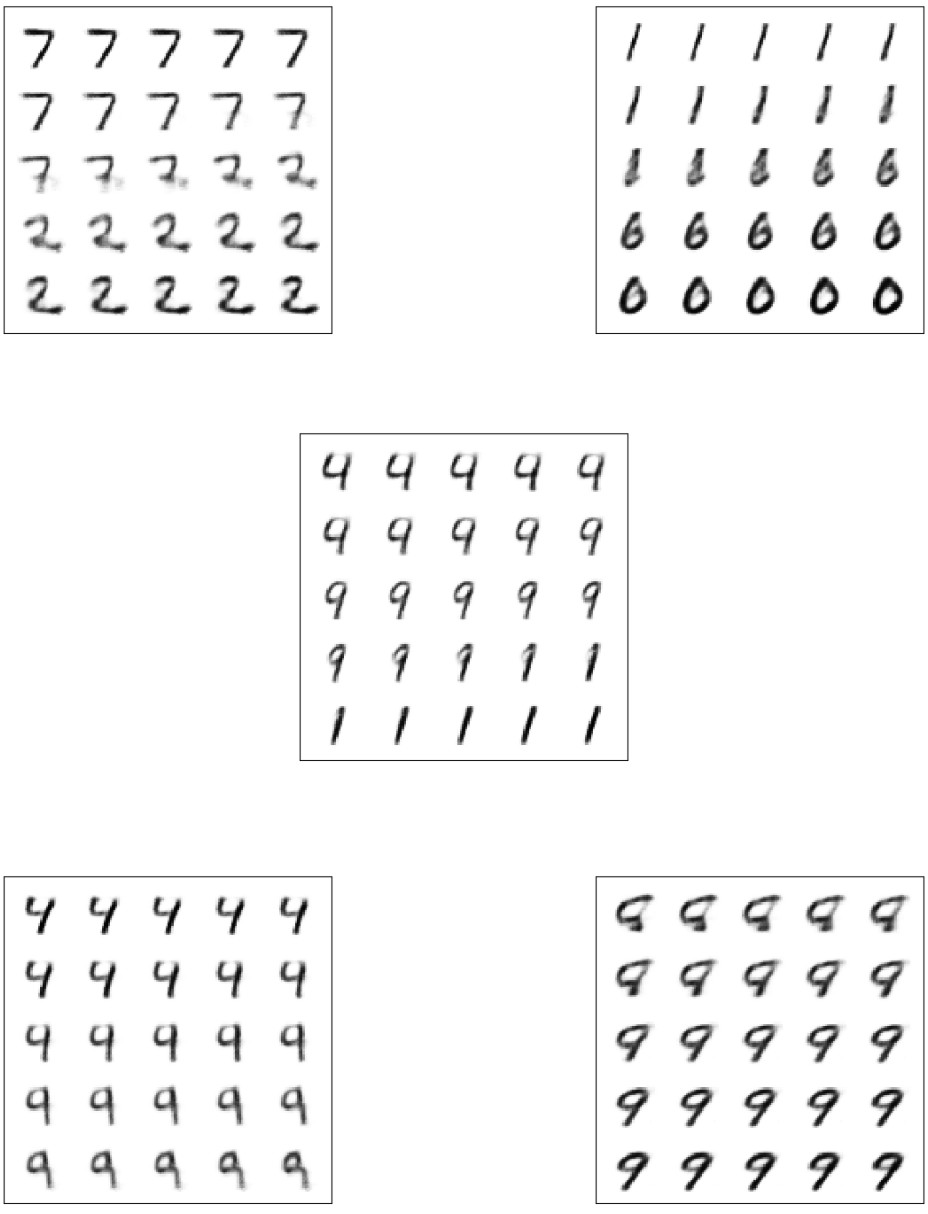

Figure 4: Five interpolations from a number to another using the segment between their latent representations.