Practical: Auto-encoding Variational Bayes

Daniel Hernández Lobato

April, 2020

1 Introduction

The goal of this practical is to familiarize you with the working principles of variational inference and to use this method for doing approximate inference in a real model. The model considered is a generative model of the form:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}\,, \tag{1}$$

where $\mathbf{x}$ is some data, $p_\theta(\mathbf{x}|\mathbf{z})$ is a conditional distribution and $p(\mathbf{z})$ is a source of noise. Figure 1 shows the corresponding probabilistic graphical model.

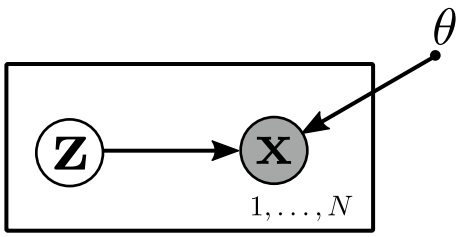


Figure 1: Probabilistic graphical model describing the generative model considered.

In this practical we will assume that the source of noise is standard Gaussian with dimension L. Namely,

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})\,. \tag{2}$$

Furthermore, we will assume that the observed data is binary, for example, a black and white image. That is, $\mathbf{x} \in \{0, 1\}^D$, where D is the dimensionality of the data. The conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$ is then assumed to be

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^{D} x_j \sigma(f_j^\theta(\mathbf{z})) + (1 - x_j)(1 - \sigma(f_j^\theta(\mathbf{z})))\,, \tag{3}$$

where $f_j^\theta(\mathbf{z})$ represents the j-th dimensional output of a deep neural network with parameters (weights and biases) θ and $\sigma(\cdot)$ is the sigmoid activation function which is given by

$$\sigma(x) = \frac{1}{1 + \exp\{-x\}}\,. \tag{4}$$

Therefore, the distribution of $\mathbf{x}$ given $\mathbf{z}$ is simply a product of D Bernoulli random variables with activation probability equal to $\sigma(f_j^\theta(\mathbf{z}))$ for $j = 1, \ldots, D$. If the conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$ is flexible enough and the dimensionality L of the latent variables $\mathbf{z}$ is big enough, we can generate

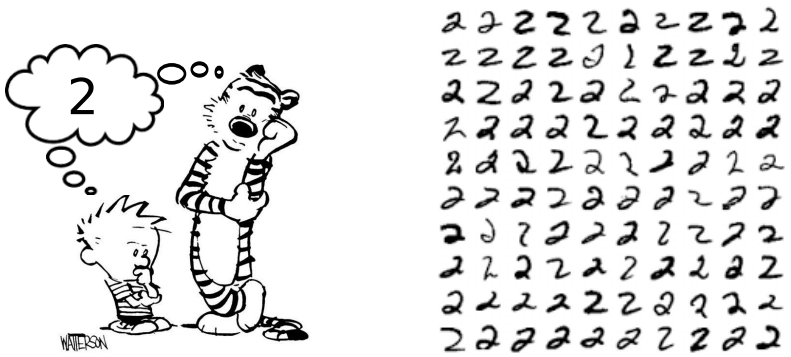


Figure 2: Illustrative process of generating some data. We first think of a high-level concept, and from that, we generate the actual data.

almost any type of data like this. For this, we only have to generate $\mathbf{z}$ from $p(\mathbf{z})$ and then use the conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$ to generate $\mathbf{x}$. You can think hence of $\mathbf{z}$ as some high-level features that contain information about the data $\mathbf{x}$. This is illustrated in Figure 2.

The task of interest is to find good parameters θ of the previous generative model to explain some observed data $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^{\mathrm{T}}$. This is, however, particularly difficult.

2 Approximate Maximum Likelihood Estimation

One approach to estimate θ given $\mathbf{X}$ is to use maximum likelihood estimation. Under this principle we simply maximize the likelihood of the observed data given θ. That is

$$\hat{\theta} = \arg\max_{\theta} \quad \log p_\theta(\mathbf{X}) = \arg\max_{\theta} \quad \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i)\,, \tag{5}$$

where we have assumed i.i.d data in $\mathbf{X}$ and where $p_\theta(\mathbf{x}_i)$ is given by (1). The problem with this approach is that computing $p_\theta(\mathbf{x}_i)$ is intractable because of the strong non-linearities of the conditional distribution $p_\theta(\mathbf{x}_i|\mathbf{z}_i)$. The consequence is that we cannot evaluate $p_\theta(\mathbf{x}_i)$ in closed form. Variational inference provides an efficient and simple way of addressing this difficulty.

The model in (1) is nothing else than the marginal likelihood of some PGM. Variational inference provides a way to approximate such a quantity. Namely, by considering the evidence lower bound. In particular,

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] + \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})|p(\mathbf{z}|\mathbf{x}))\,, \tag{6}$$

where

$$\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})|p(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} d\mathbf{z} \geq 0\,. \tag{7}$$

Therefore, the difference between $\log p(\mathbf{x})$ and the first term in the r.h.s. of (6) is simply the Kullback-Leibler divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and the actual posterior $p(\mathbf{z}|\mathbf{x})$. Because this divergence is always positive or equal to zero, we have that

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \mathcal{L}(\mathbf{x}, \theta, \phi)\,, \tag{8}$$

Furthermore, by maximizing with respect to ϕ, the parameters of $q_\phi(\mathbf{z}|\mathbf{x})$, we will effectively be minimizing $\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})|p(\mathbf{z}|\mathbf{x}))$. This means that $q_\phi(\mathbf{z}|\mathbf{x})$ can be understood as a distribution

approximating $p(\mathbf{z}|\mathbf{x})$. This distribution will be implemented in practice by a factorizing Gaussian whose parameters are given by the output of a deep neural network. That is,

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{j=1}^{L} \mathcal{N}(z_j|\mu_j^\phi(\mathbf{x}), \nu_j^\phi(\mathbf{x}))\,, \tag{9}$$

where $\mathcal{N}(z_j|\mu_j^\phi(\mathbf{x}), \nu_j^\phi(\mathbf{x}))$ denotes the p.d.f. of a Gaussian distribution with mean $\mu_j^\phi(\mathbf{x})$ and variance[1] $\nu_j^\phi(\mathbf{x})$.

Importantly, if $\mathcal{L}(\mathbf{x}, \theta, \phi)$ is maximized with respect to ϕ, one should expected that $\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})|p(\mathbf{z}|\mathbf{x}))$ is very small. In this setting, $\mathcal{L}(\mathbf{x}, \theta, \phi) \approx \log p_\theta(\mathbf{x})$. Therefore, maximizing $\mathcal{L}(\mathbf{x}, \theta, \phi)$ with respect to θ is expected to increase $\log p_\theta(\mathbf{x})$. This is illustrated in Figure 3.

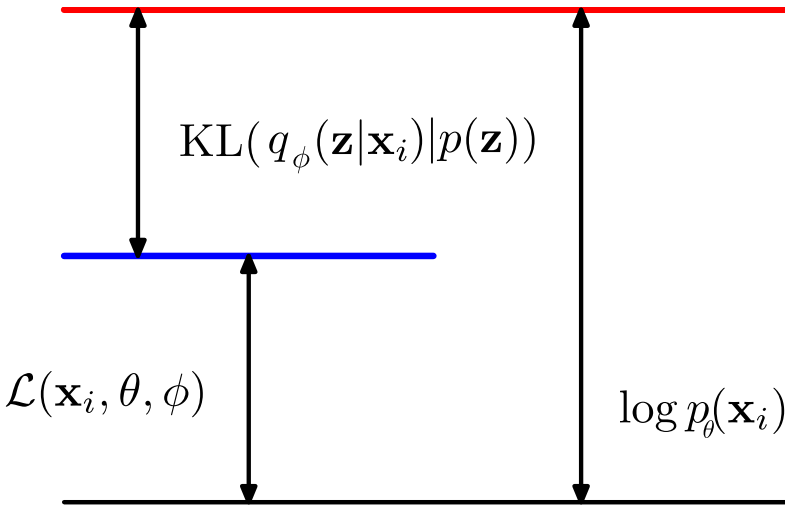


Figure 3: Decomposition of the marginal likelihood $\log p(\mathbf{x}_i)$ employed by variational inference.

Variational inference can hence be used to find good model parameters θ for the generative model described in (1) simply by maximizing with respect to θ and ϕ

$$O(\phi, \theta) = \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i, \theta, \phi) = \sum_{i=1}^{N} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log \frac{p_\theta(\mathbf{x}_i|\mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x}_i)} \right]. \tag{10}$$

This is a lower bound on the log marginal likelihood of the data. That is $\log p_\theta(\mathbf{X}) \geq O(\phi, \theta)$. An un-expected benefit is that we obtain for free a recognition model $q_\phi(\mathbf{z}|\mathbf{x})$ that can be used to infer the latent variable $\mathbf{z}$ that was used to generate the data point $\mathbf{x}$. The combination of the generative model $p_\theta(\mathbf{x}|\mathbf{z})$ and the recognition model $q_\phi(\mathbf{z}|\mathbf{x})$ is known as a variational auto-encoder VAE. This model is described graphically in Figure 4. Note that both models are implemented in practice using deep neural networks.

3 Optimization of the Objective

The objective in (10) that needs to be optimized under the VAE can be written like:

$$O(\phi, \theta) = \sum_{i=1}^{N} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log p_\theta(\mathbf{x}_i|\mathbf{z}) \right] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i)|p(\mathbf{z}))\,, \tag{11}$$

[1] In the implementation it is more convenient that the neural network outputs the logarithm of the variance or the logarithm of the standard deviation, to avoid having to constrain the variance to be strictly positive.

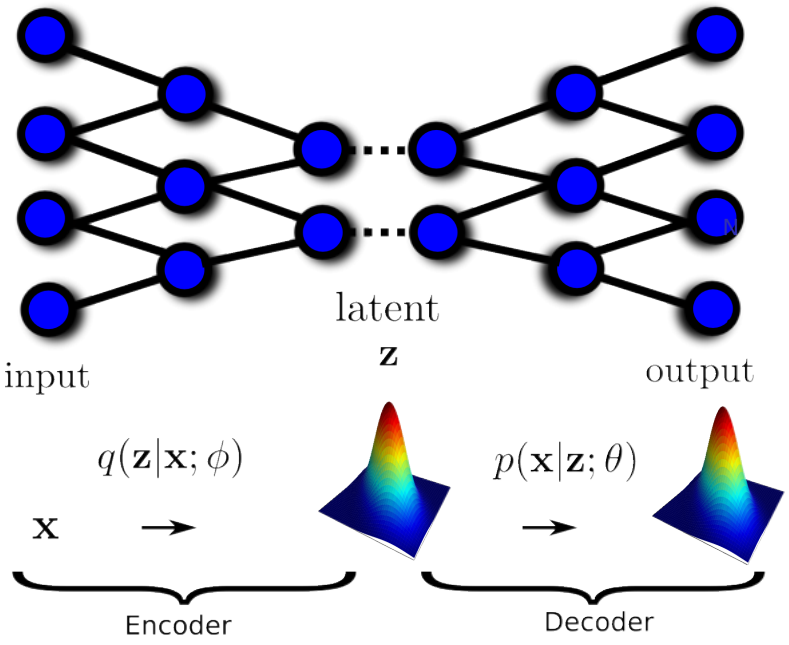


Figure 4: Illustration of a variational auto-encoder in which the $\mathbf{x}$ are assumed to take real values.

where $\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i)|p(\mathbf{z})$ is the Kullback-Leibler divergence between the posterior approximation $q_\phi(\mathbf{z}|\mathbf{x}_i)$ and the prior $p(\mathbf{z})$. Because these two distributions are Gaussian, this expression has a closed form. Namely,

$$\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i)|p(\mathbf{z})) = \sum_{j=1}^{L} \frac{1}{2} \left(\nu_j^\phi(\mathbf{x}_i) + \mu_j^\phi(\mathbf{x}_i)^2 - 1 - \log \nu_j^\phi(\mathbf{x}_i) \right) \tag{12}$$

where we have used the fact that the prior $p(\mathbf{z})$ is a standard Gaussian distribution.

When the number of training points is very large, the objective $O(\phi, \theta)$ can be very expensive to compute. It is better to approximate it using a mini-batch of data points $\mathcal{B}$. Namely,

$$O(\phi,\theta) \approx \tilde{O}(\phi,\theta) = \frac{N}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log p_\theta(\mathbf{x}_i|\mathbf{z}) \right] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i)|p(\mathbf{z})) \,, \tag{13}$$

where the factor $N/|\mathcal{B}|$ has been included to account for the fact that we are sub-sampling the training data. We said before that the KL term has a closed form expression. Therefore, we can easily evaluate that term. The problem comes when we have to evaluate the first term that depends on $p(\mathbf{x}_i|\mathbf{z})$. That is, $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log p_\theta(\mathbf{x}_i|\mathbf{z}) \right]$. This expectation has no closed-form solution, and will have to be approximated.

3.1 Reparametrization Trick

To approximate $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log p_\theta(\mathbf{x}_i|\mathbf{z}) \right]$ (and also its gradients), we will use the reparametrization trick. This trick consists in approximating the previous expression by Monte Carlo. That is,

$$\tilde{O}(\phi,\theta) \approx \tilde{\tilde{O}}(\phi,\theta) = \frac{N}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \log p_\theta(\mathbf{x}_i|\mathbf{z}^i) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i)|p(\mathbf{z})) \,, \tag{14}$$

where $\mathbf{z}^i$ has been generated from $q_\phi(\mathbf{z}|\mathbf{x}_i)$. Note that in this case we are considering a single Monte Carlo sample per each data point $\mathbf{x}_i$ in the batch $\mathcal{B}$. Importantly, we can now separate

the randomness in each $\mathbf{z}^i$ from the parameters of $q_\phi(\mathbf{z}|\mathbf{x}_i)$, which will depend on ϕ. For this, we simply have to set

$$z_j^i = \mu_j^\phi(\mathbf{x}_i) + \sqrt{\nu_j^\phi(\mathbf{x}_i)}\epsilon_i^j \,, \tag{15}$$

where $\epsilon_i^j \sim \mathcal{N}(0,1)$. The consequence is that we can simply obtain a noisy estimate of the gradients of the actual objective $O(\theta,\phi)$ with respect to θ and ϕ by computing the gradients of $\tilde{\tilde{O}}(\phi,\theta)$ with respect to those parameters. That is,

$$\frac{\partial O(\theta,\phi)}{\partial\theta} \approx \frac{\partial\tilde{\tilde{O}}(\theta,\phi)}{\partial\theta} \,, \qquad\qquad \frac{\partial O(\theta,\phi)}{\partial\phi} \approx \frac{\partial\tilde{\tilde{O}}(\theta,\phi)}{\partial\phi} \,. \tag{16}$$

Importantly, these gradient approximations are unbiased, which means that they coincide with the actual gradient in expectation. Furthermore, the previous gradients can be obtained automatically using software such as auto-grad (available at https://github.com/HIPS/autograd).

3.2 Stochastic Optimization

The fact that we can easily obtain a noisy estimate of the gradient of the objective allows for using stochastic optimization tools such as the ADAM algorithm for optimizing the lower bound on the log marginal likelihood. ADAM is straightforward to implement and is computationally efficient. The ADAM algorithm is displayed in Figure 5. This algorithm only requires access to a noisy estimate of the gradients of the objective.

Algorithm 1: g_t indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1-\beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1-\beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 5: The ADAM algorithm for optimizing stochastic objectives.

4 MNIST Dataset

In the code provided the generative model is trained on the MNIST dataset. This is a dataset of 28×28 images in gray-scale describing hand-written digits. The number of data instances are 60,000 training images and 10,000 test images. Some examples of the images contained in this dataset are displayed in Figure 6.

The model described considers only binary data. That is, data taking values in the set $\{0, 1\}$. Notwithstanding, the pixels of the MNIST dataset images take values in the interval $[0, 1]$. For this reason, the data has to be binarized when training the model. In the code provided this step is implemented by generating a uniform random variable u in the interval $[0, 1]$. If the pixel intensity is bigger than u, we set the pixel value equal to 1 and 0 otherwise.

In the code provided, the images of the MNIST dataset are stored as vectors of size $28 \times 28 = 784$ entries. The data is also automatically loaded. No action is required for this.

Figure 6: Sample images extracted from the MNIST dataset.

5 Tasks

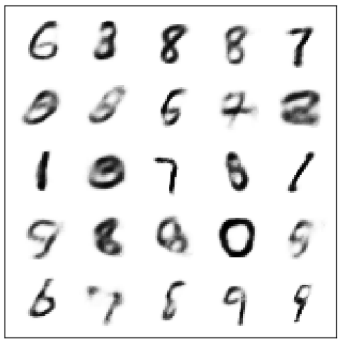
This section describes the different tasks that you have to do in this practical. To facilitate the work to be done, you will start from a partially completed code, implementing a VAE to be trained on the MNIST dataset. This code is found in the file *vae.py*. The code can be run simply by typing *python vae.py*. Write a brief report summarizing the results obtained.

- **Task 1**: Complete the missing parts of the code to allow for training a VAE on the MNIST dataset. For this, you will have to complete the following functions:
  - *sample_latent_variables_from_posterior()*
  - *bernoulli_log_prob()*
  - *compute_KL()*
  - *vae_lower_bound()*

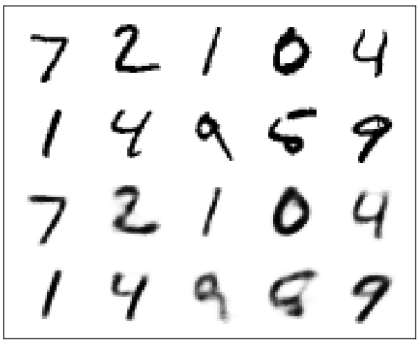
  Take a look at the code in *vae.py* to get further information about what each function should do. Ask the teacher for details if you have any question.

- **Task 2**: Complete the initialization of the ADAM parameters and write the ADAM updates in the main training loop of the code provided in *vae.py*. Look into the code for further details.

- **Task 3**: After training (this can take a few minutes) you should use the generated model to carry out these sub-tasks (do some testing with a small number of epochs to check that it works fine):
  - **Subtask 3.1**: Generate 25 images from the generative model. This should be done by drawing $\mathbf{z}$ from the prior, to then generate $\mathbf{x}$ using the conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$. Set the pixel intensity of the image equal to the activation probability. Do not binarize the images. Save the images to a file using the function *save_images*. You should obtain something similar to what appears in the image below.

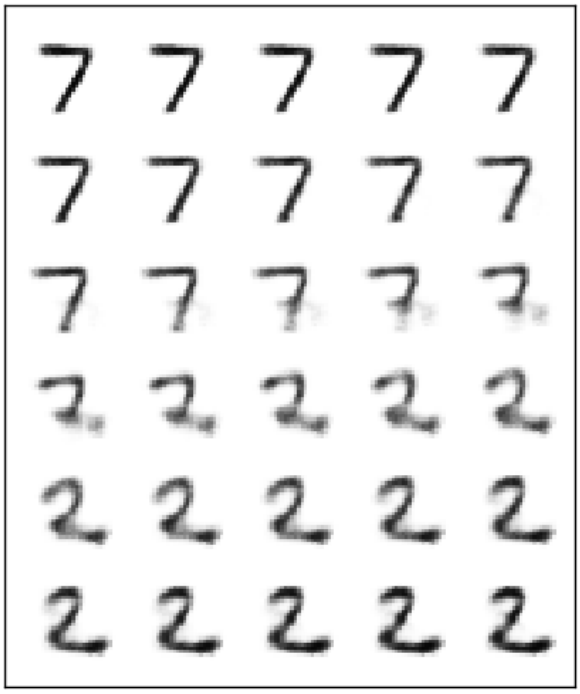


  - **Subtask 3.2**: Generate 10 image reconstructions using the recognition model and then the generative model. Choose the first 10 images from the test set. The reconstructions are obtained by generating $\mathbf{z}$ using $q_\phi(\mathbf{z}|\mathbf{x})$ and then generating $\mathbf{x}$ again using $p_\theta(\mathbf{x}|\mathbf{z})$. Set the pixel intensity of the image equal to the activation probability. Do not binarize the images. Save the images to a file using the function *save_images*. You should obtain something similar to what appears in the image below (first two rows original images, second two rows reconstructions).



  - **Subtask 3.3**: Generate 5 interpolations in the latent space from one image to another. Consider the first and second image in the test set, the third and fourth image in the test set and so on. The interpolations should be obtained by finding the latent representation of each image. Consider only the mean of the predictive model $q(\mathbf{z}|\mathbf{x})$ as the latent representation. Ignore the variance. Assume that you would like to interpolate between the latent representation of the first image $\mathbf{z}_1$ and the latent representation of the second image $\mathbf{z}_2$. The interpolation is obtained by computing $\mathbf{z}^s_{\text{mix}} = \mathbf{z}_1 s + (1-s)\mathbf{z}_2$ for $s \in [0, 1]$. Consider a grid of 25 values in the interval $[0, 1]$. Given each $\mathbf{z}^s_{\text{mix}}$, generate the corresponding image using $p_\theta(\mathbf{x}|\mathbf{z})$. Set the pixel intensity of the image equal to the activation probability. Do not binarize the images. Save the images to a file using

the function *save_images*. You should obtain something similar to what appears in the image below (for the first and second images in the test set).



6 References

- Kingma, D. P., & Ba, J. Adam: A method for stochastic optimization, International Conference on Learning Representations, 2014.
- Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes, International Conference on Learning Representations, 2014.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278-2324, 1998.