Homework 2 – Generative AI (CS/DS 552, Whitehill, Spring 2025)

Collaboration policy: You may complete this assignment with a partner, if you choose. In that case, both partners should sign up on Canvas in a **pre-made group** as a team, and only one of you should submit the assignment. You are permitted to use ChatGPT (or another AI tool) without restriction to help save you time typing boilerplate code, making complicated visualizations, and overcoming tedious syntactic issues. However, *you must fully understand all the work that you submit.*

1. **(Variational) Auto-Encoders for Face Image Generation [50 pts]**

   In this exercise you will implement and train a (V)AE to generate face images of size 24×24 pixels. To get started, first download `faces_vae.npy` from Canvas.

   (a) **VAE Training & Data Generation**: Build and train either a convolutional or fully-connected (either is fine) VAE. I recommend using Google Colab with free (but limited) GPU support, e.g., T4 GPU. To get satisfactory results (i.e., the faces will still look a bit blurry or have some minor artifacts, but they should be clearly recognizable as faces and look more realistic than with P-PCA), you shouldn't need more than 30minutes of training. Visualize 100 generated faces as a 10×10 collage. Implementation hints: (1) Before training, normalize the pixel values of the training data to be in $[0, 1]^m$. (2) While it is possible to train a decoder (as part of the VAE) that estimates $P(\mathbf{x} \mid \mathbf{z})$ as a Gaussian distribution (corresponding to an identity activation function), you can often get better training performance if you instead use a logistic sigmoid activation function to parameterize a Bernoulli distribution, i.e., the output of the decoder $\mu_{\mathbf{x}}$ is constrained to lie in $[0, 1]^m$. In this case, instead of `MSELoss`, you can train with `BCELoss`, and everything else in the VAE stays the same. (3) You may need to use the β-VAE method shown in class and try various values for $\beta < 1$. **[25 pts]**

   (b) **AE Training & Data Generation**: Using the exact same architecture as part (a), train a *standard* auto-encoder (AE) just with a reconstruction loss (but no KL-divergence term), and without random sampling in the latent space. Visualize 100 generated faces as a 10×10 collage. **[10 pts]**

   (c) **Comparing $Q(\mathbf{z} \mid \mathbf{x})$ between VAE and AE**: Based on the training data $\{\mathbf{x}^{(i)}\}$, visualize the distribution $Q(\mathbf{z} \mid \mathbf{x})$ of latent codes $\{\mathbf{z}^{(i)}\} \subset \mathbb{R}^d$ for the (i) AE and (ii) VAE. (To visualize these distributions, you can use PCA to reduce the dimensionality of the latent space to 2 dimensions.) How do these distributions compare between the two models, and why? **[6 pts]**

   (d) **Sampling linearly in the latent space**: Randomly pick two latent codes $\mathbf{z}, \mathbf{z}' \in \mathbb{R}^d$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Then, pick 20 points equally spaced along the line segment between $\mathbf{z}$ and $\mathbf{z}'$. For each such point $\mathbf{z}^{(i)}$, use the trained VAE decoder to generate a face image $\mathbf{x}^{(i)} = \mathbb{E}[\mathbf{x} \mid \mathbf{z}^{(i)}]$ (which is just the output $\mu_{\mathbf{x}}$ of the decoder). Display all 20 of them in a collage. Describe any pattern you notice in how the generated faces change as a function of the linearly sampled latent codes they were generated from. **[3 pts]**

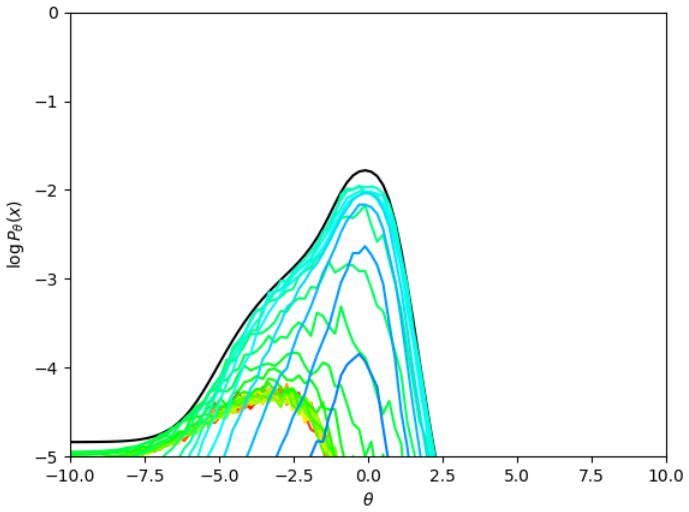
   (e) **Comparing $P(\mathbf{x} \mid \mathbf{z})$ between VAE and P-PCA**: Building on part (c), visualize each generated $\mathbf{x}^{(i)}$ as a vector by projecting it into a 2-d space (use PCA trained on the entire training set to reduce the dimensionality). In your plot, display a number ($i = 1, 2, \ldots, 20$) beside each generated vector $\mathbf{x}^{(i)}$. Do this for both (i) the trained VAE and (ii) the P-PCA model from Homework 1 (you can re-train it on `faces_vae.npy` for a fair comparison). How does the pattern of the generated faces differ between these two models, and why? **[6 pts]**

2. **Feel Your ELBO [20 pts]**

   Maximizing $\log P_\theta(\mathbf{x}) = \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z}$ w.r.t. θ in a VAE is, in general, computationally intractable since this integral has no closed-form solution and a numeric integration encounters the curse of dimensionality. That is why we instead maximize the Evidence Lower BOund (ELBO) – finding values for ϕ, θ (of the encoder and decoder, respectively) that are good for the ELBO helps us to find

a θ that is good for $\log P_\theta(\mathbf{x})$ itself. However, for tiny latent dimension (say, $d = 1$, in which case z is just a scalar), we can numerically compute this integral in "brute-force" fashion. In this exercise, you will plot both the true log-likelihood as well as the ELBO over the course of training and show how the ELBO and the parameter estimates for ϕ, θ evolve.

This is an open-ended exercise, and the purpose is to help you get better intuition for how maximizing the ELBO w.r.t. θ and ϕ can maximize the true log-likelihood that we actually care about. Your goal is to create a simulation where you define a VAE for some generation task (it could be anything, and it can be very simple, e.g., 1-d data), show the true log-likelihood (computed via brute-force), and then show how the ELBO can approximate this function tightly. The figure below shows the kind of solution we are looking for – the horizontal axis represents θ, the black curve is $\log P(\mathbf{x})$ (true log-likelihood), and each of the other curves is an approximation for a different value of ϕ. Here, each of θ and ϕ was, for simplicity (and to enable visualization on a 2-d plane) just a scalar. Crucially, you cannot just make up some fake curves – *they must actually represent the* $\log P(\mathbf{x})$ *and the ELBO or a real (but simple) VAE.* Thinking through what these functions mean and how to generate them will help you to understand what VAEs are and how they are optimized – it certainly did for me.



Submission: Create a Zip file containing all your Python code (which can be in multiple files) and your PDF file, and then submit on Canvas. If you are working as part of a group, then only **one** member of your group should submit (but make sure you have already signed up in a pre-allocated team for the homework on Canvas).