Computer Vision I: Project 5 (10 points)

Due on Dec. 31th, 11:59pm

1 Background

This project is based on **Section 11.2.3: Learning by alternating back-propagation**. Read the textbook for more information.

1.1 Objective

The objective of this project is to experience the multi-layered generator network, which is a top-down generative model. It is a nonlinear generalization of the factor analysis model. This top-down generative model has the following properties:

1. Embedding. The model embeds the high-dimensional non-Euclidean manifold formed by the observed examples into the low-dimensional Euclidean space of the latent factors so that linear interpolation in the low-dimensional factor space results in nonlinear interpolation in the data space.
2. Analysis. The model disentangles the variations in the observed examples into independent variations of latent factors.
3. Synthesis. In contrast to the bottom-up descriptive model which samples in high-dimensional space to synthesize examples, the top-down model can synthesize new examples by sampling the factors from the known low-dimensional prior distribution and transforming the factors into the synthesized examples.

1.2 Python Library

Please install the latest matplotlib, numpy, pillow, torch and torchvision. You are also welcome to utilize any libraries of your choice, **but please report them in your report (for autograder)! Again, report any customized library in the report (do not go too crazy as this will add a significant burden to TAs).**

1.3 What to hand in?

Please submit both a formal report and the accompanying code. For the report, kindly provide a PDF version. You may opt to compose a new report or complete the designated sections within this document, as can be started by simply loading the tex file to Overleaf. Your score will be based on the quality of **your results**, **the analysis** (diagnostics of issues and comparisons) in your report, and your **code implementation**. You may delete all the images before handing them in, as they may be too large for the autograder.

Notice. Do not modify the function names, parameters, and returns in the given code, unless explicitly specified in this document.

1.4 Help

Make a diligent effort to address any encountered issues independently, and in cases where challenges exceed your capabilities, do not hesitate to seek assistance! Collaboration with your peers is permitted, but it is crucial that you refrain from directly examining or copying one another's code. Please be aware that you'll fail the course if our **code similarity checker**, which has found some prohibited behaviors before, detects these violations. For details, please refer to: https://yzhu.io/s/teaching/plagiarism.

2 Introduction to the Deep Generative Model

Notations. Let $\mathbf{I}$ be a D-dimensional observed example, such as an image. Let z be the d-dimensional vector of continuous latent factors, $z = (z_k, k = 1, ..., d)$. The traditional factor analysis model is $\mathbf{I} = Wz + \epsilon$, where W is $D \times d$ matrix, ϵ is a D-dimensional error vector or the observational noise, usually $d < D$, $z \sim \mathrm{N}(0, I_d)$, and $\epsilon \sim \mathrm{N}(0, \sigma^2 I_D)$. The deep generative model generalizes the linear mapping Wz to a non-linear mapping $g(z; \theta)$, where g is a ConvNet, and θ collects all the connection weights and bias terms of the ConvNet. Then the model becomes

$$\begin{aligned} &\mathbf{I} = g(z; \theta) + \epsilon, \\ &z \sim \mathrm{N}(0, I_d), \ \epsilon \sim \mathrm{N}(0, \sigma^2 I_D), \ d < D. \end{aligned} \tag{1}$$

The reconstruction error is $\|\mathbf{I} - g(z; \theta)\|^2$.

Although $g(z; \theta)$ can be any nonlinear mapping, the ConvNet parameterization of $g(z; \theta)$ makes it particularly close to the original factor analysis. Specifically, we can write the top-down ConvNet as follows:

$$z^{(l-1)} = g_l(W_l z^{(l)} + b_l), \tag{2}$$

where g_l is element-wise nonlinearity at layer l, W_l is the weight matrix, b_l is the vector of bias terms at layer l, and $\theta = (W_l, b_l, l = 1, ..., L)$. $z^{(0)} = g(z; \theta)$, and $z^{(L)} = z$. The top-down ConvNet (2) can be considered a recursion of the original factor analysis model, where the factors at the layer $l - 1$ are obtained by the linear superposition of the basis vectors or basis functions that are column vectors of W_l, with the factors at the layer l serving as the coefficients of the linear superposition.

We employ the alternating back-propagation algorithm for maximum likelihood learning of the generator network that iterates the following two steps:

(1) *Inferential back-propagation*: For each training example, infer the continuous latent factors by Langevin dynamics.

(2) *Learning back-propagation*: Update the parameters given the inferred latent factors by gradient descent.

Specifically, the joint density is $p(z, \mathbf{I}; \theta) = p(z)p(\mathbf{I}|z; \theta)$, and

$$\log p(z, \mathbf{I}; \theta) \quad = \quad -\frac{1}{2\sigma^2}\|\mathbf{I} - g(z; \theta)\|^2 - \frac{1}{2}\|z\|^2 + \text{constant}, \tag{3}$$

where the constant term is independent of z and θ.

For the training data $\{\mathbf{I}_i, i = 1, ..., n\}$, the generator model can be trained by maximizing the log-likelihood

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p(\mathbf{I}_i; \theta). \tag{4}$$

The gradient of $L(\theta)$ is obtained according to the following identity

$$\frac{\partial}{\partial \theta} \log p(\mathbf{I}; \theta) = \mathrm{E}_{p(z|\mathbf{I};\theta)}\left[\frac{\partial}{\partial \theta} \log p(z, \mathbf{I}; \theta)\right]. \tag{5}$$

In general, the expectation in (5) is analytically intractable and has to be approximated by MCMC that samples from the posterior $p(z|\mathbf{I}; \theta)$, such as the Langevin inference dynamics, which iterates

$$z_{\tau+1} = z_\tau + \frac{s^2}{2}\frac{\partial}{\partial z} \log p(z_\tau, \mathbf{I}; \theta) + se_\tau, \tag{6}$$

where τ indexes the time step, s is the step size, and e_τ denotes the noise term, $e_\tau \sim \mathrm{N}(0, I_d)$. for each z_i, only a single copy of z_i is sampled from $p(z_i|\mathbf{I}_i, \theta)$ by running a finite number of steps of Langevin dynamics starting from the current value of z_i, i.e., the warm start. With z_i sampled from $p(z_i \mid \mathbf{I}_i, \theta)$ for each observation $\mathbf{I}_i$ by the Langevin inference process, the Monte Carlo approximation to $L'(\theta)$ is

$$\begin{aligned} L'(\theta) &\approx \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial \theta} \log p(z_i, \mathbf{I}_i; \theta) \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\sigma^2}\left(\mathbf{I}_i - g(z_i;\theta)\right) \frac{\partial}{\partial \theta} g(z_i;\theta). \end{aligned} \tag{7}$$

The updating of θ solves a nonlinear regression problem so that the learned θ enables better reconstruction of $\mathbf{I}_i$ by the inferred z_i.

3 Experiment

For the lion-tiger images, you will learn a generative model with 2-dim latent factor vector. To reduce the computational complexity, all images can be resized into the size of 128×128 pixels.

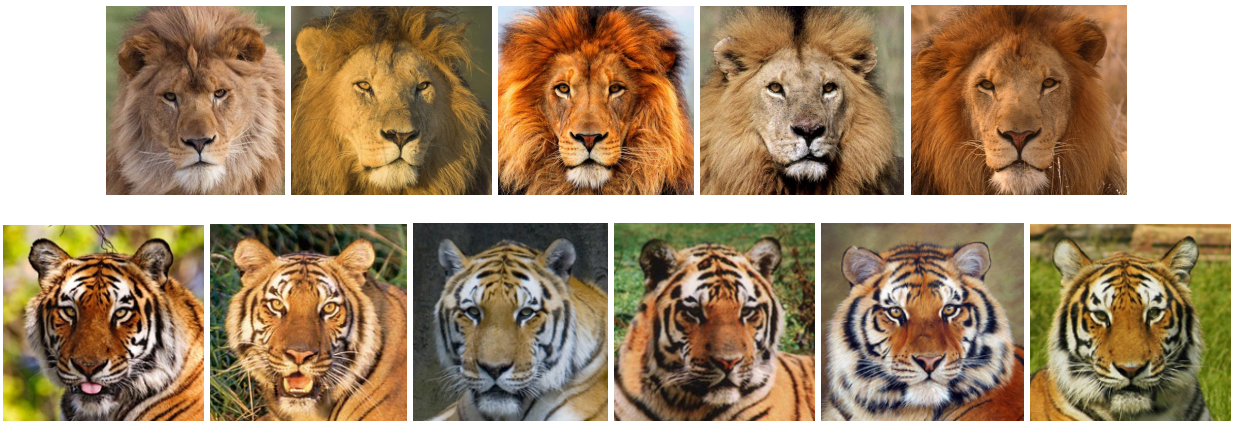

Figure 1: lion-tiger images.

Fill the blank part of `abp.py`. Design your own structure of the generator. To generate vivid results, you should adjust the learning rate, discretization step size of the Langevin dynamics, the number of Langevin steps, and the annealing or tempering parameter σ^2 in Eq.3.

4 Submission

(1) Reconstructed images of training images, using the inferred z from training images. Try both the warm-start scheme (running a finite number of steps of Langevin dynamics starting from the current value of z_i) and cold-start scheme (running a finite number of steps of Langevin dynamics starting from the fixed prior distribution of the latent factors)

(2) Randomly generated images, using randomly sampled z.

(3) Generated images with linearly interpolated latent factors from (-d, d). For example, you interpolate 12 points from (-d, d) for each z dimension. Then you will get a 12×12 panel of images. Determine the

reasonable range (-d, d), and you should be able to see that tigers slightly change to lions.

(4) Plot of loss over iteration.

Write a report to show your results. And zip the report with your code.

Figure 2: warm - cold images of 1000 epochs and 2000 epochs

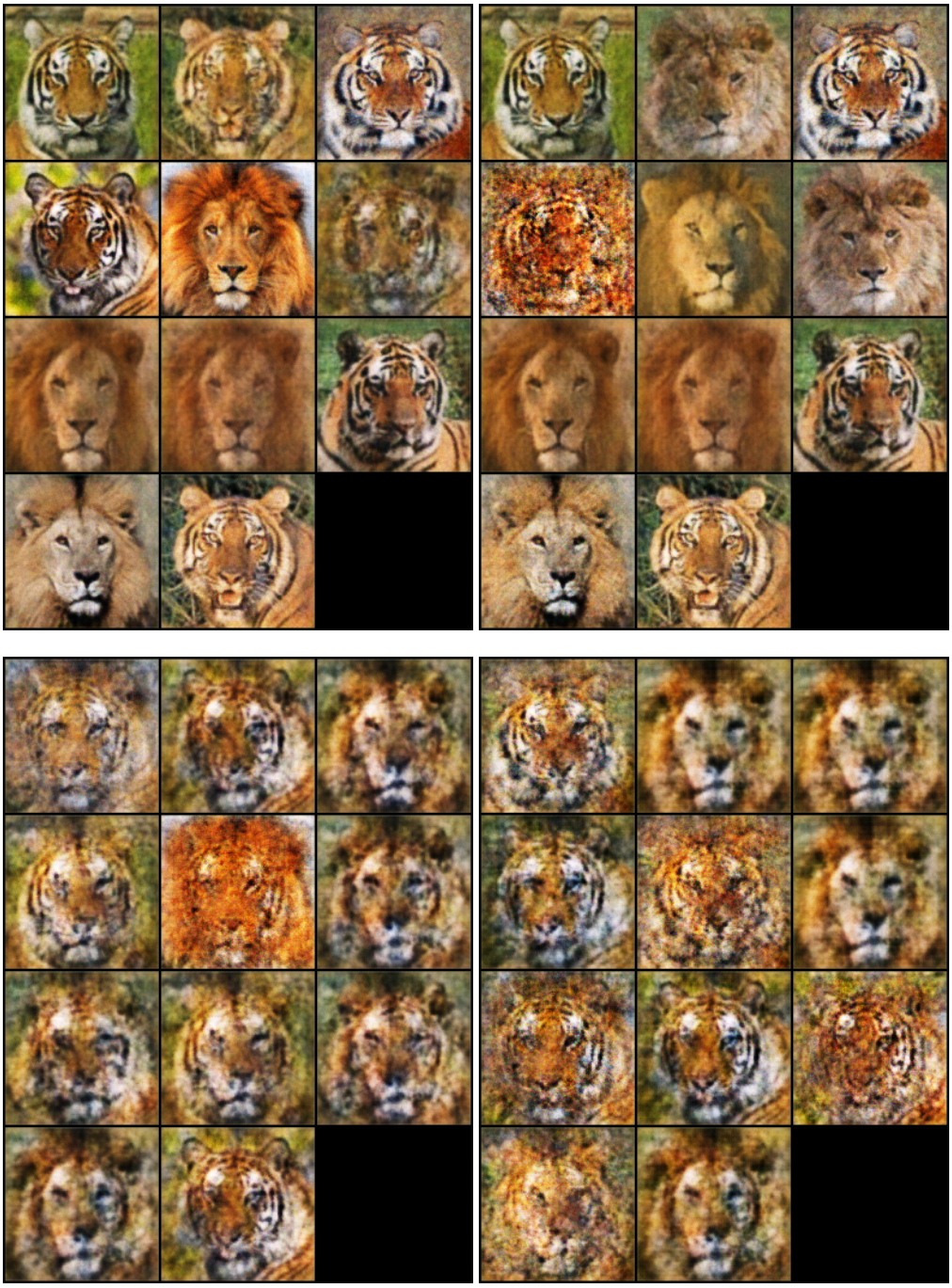

Figure 3: warm - cold randomly generated images

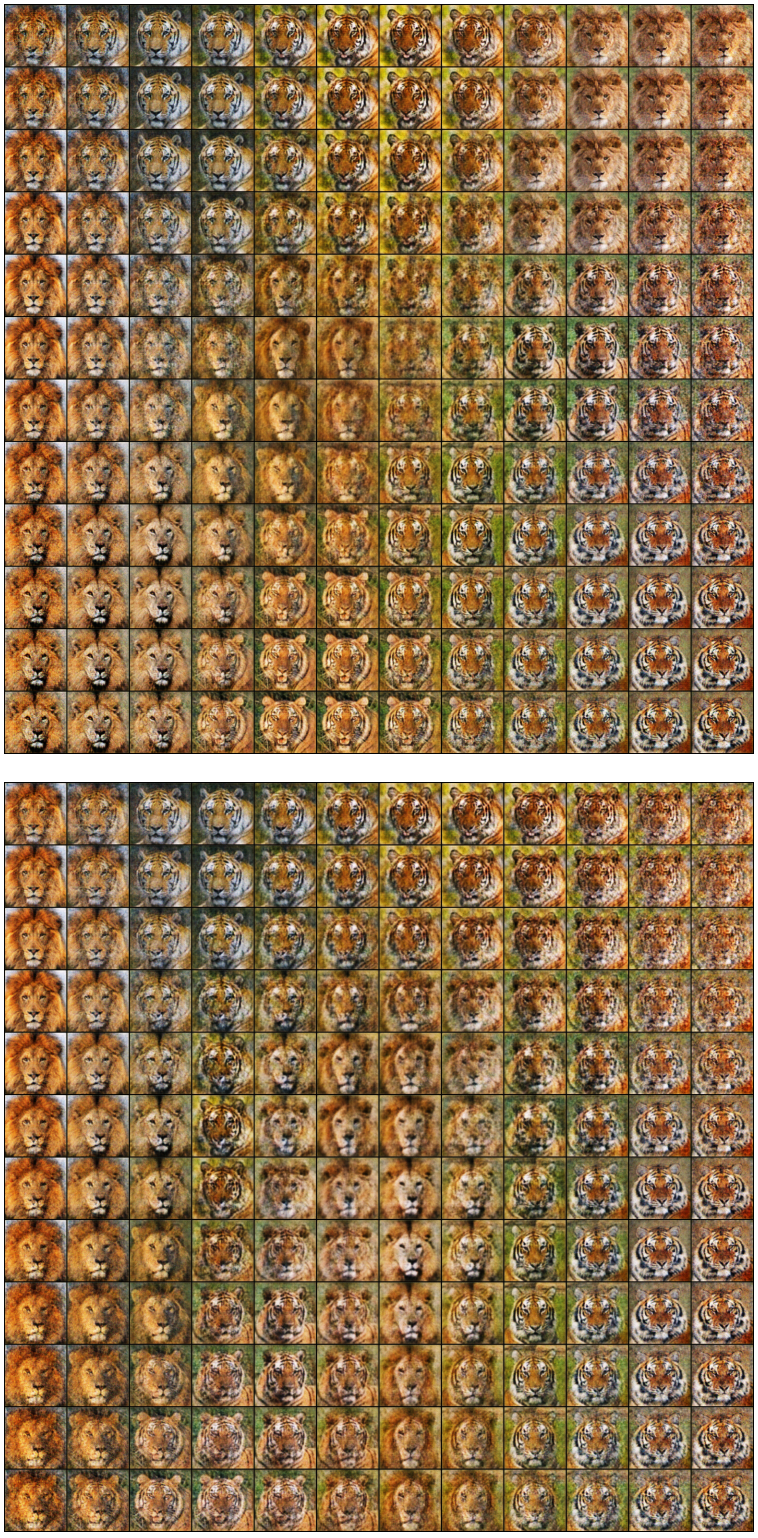

Figure 4: warm - cold interpolated images

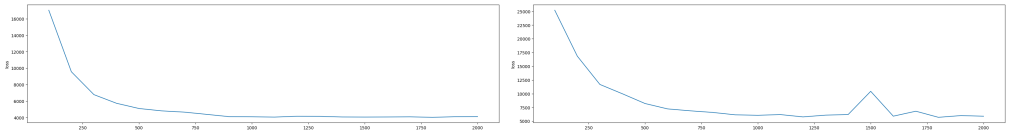

Figure 5: warm - cold loss