CIS680: Vision & Learning
Assignment 4: Deep Generative Models
Due: Dec. 17, 2017 at 11:59 pm

Instructions

- This is a **Group** assignment. You should team up with the same teammate(s) as the final project. One group turns in exactly one copy of the report and code to avoid confusion. Include all the names of team members at the beginning of the report.
- This assignment compliments the final project. Both will contribute to your final grade. However, you should finish at least any one part of this assignment no matter how big your final project is.
- All the code should be written in Python. You are welcome to use Tensorflow or PyTorch to complete this homework.
- You must submit your solutions online on **Canvas**. Compress your files into a ZIP file named "4_<penn_key>.zip", which should contain **1 PDF report** and **3 folders containing the Python code for each part**. Note that you should include all the figures in your report.

Overview

This homework aims at implementing, analyzing, and tuning/improving deep generative models. Deep generative models has emerged to be one of the promising directions in both generative models and deep learning. Recent success in deep generative model has shown the potential to generate high resolution real-looking images [10]. (Check out their video.) In this assignment, you will implement the two major branches of deep generative models, i.e., Variational Autoencoders (VAEs) and generative adversarial networks (GANs). Unlike previous assignments where architectures are specified, you only get hints and have to

(a) CUFS Sketches Dataset for part 1 & 2

(b) CelebA Dataset for part 1 & 2

(c) CUFS Sketches Dataset for part 3

Figure 1: Sample images of each dataset. (a) Cropped, resized, and binarized sketches from the CUFS dataset. (b) Cropped and resized grayscale images from the CelebA dataset. (c) Sketch-photo pair images from the CUFS dataset.

come out the architectures on your own in this assignment.

This homework consists of three parts.

1. Implement and analyze Variational Autoencoders (VAEs) [11].
2. Implement and analyze generative adversairal networks (GANs) [7].
3. Implement and play with image-to-image translation or pix2pix [9].

Datasets

We mainly use two datasets in this assignment: CUHK Face Sketch Database (CUFS) [3] and Large-scale CelebFaces Attributes (CelebA) [1] Datasets.

CUFS serves as a testbed where you can experiment with your architectures. CelebA will provide you a rigid result once you upscale the architectures.

For both datasets, we provide processed images on the course website. All images are cropped and resized to 64×64.

The images from the datasets are shown in Fig. 1.

General Guidelines for Architecture Design

As described in the Overview, you will design your own architectures in this assignment. Here are some guidelines as a starting point:

- Always normalize the images so that their values are $\in [-1, 1]$ unless specified otherwise.
- It is better to set the numbers of channels as powers of 2.
- Use batch normalization after each convolutional layer.
- When building encoders, use convolutional layers with kernel size 3 and stride 2 as downsampling. Double the number of channels right before downsampling.
- When building decoders, use transpose convolutional layers or resize the activation maps as upsampling. Halve the number of channels after upsampling.

  In Tensorflow, use functions such as tf.nn.conv2d_transpose() or tf.image.resize_nearest_neighbor().
- When building generators, use regular ReLU activations except for the last layer with Tanh activations.
- When building discriminators, use leaky ReLU ($\alpha = 0.2$) activations.
- Use Adam optimizer with $\beta_1 = 0.5$. Start with learning rates between 10^{-3} and 10^{-4}.
- Train models with $2,000$ to $5,000$ iterations. Set the batch size as 50 or 64, up to 128, depending on which dataset you use.
- Use a narrow network (number of channels ≤ 128) for the CUFS dataset and enlarge it (number of channels ≤ 1024) for the CelebA dataset.

Note that these guidelines mainly serve as a starting point for your architecture design. You are free to (and need to) explore the architectures and hyper-parameters.

1 Variational Autoencoders (35%)

Recently, Variational Autoencoders (VAEs) [11, 6] have emerged as one of the most popular approaches to unsupervised learning of complicated distributions. Before starting this part, we encourage you to read at least one of the two papers [11, 6].

The major difference between Autoencoders and VAEs is that Autoencoders compress the data while VAEs learn the probability distribution of the data. As a result, we can sample from the distribution and generate new images using VAEs.

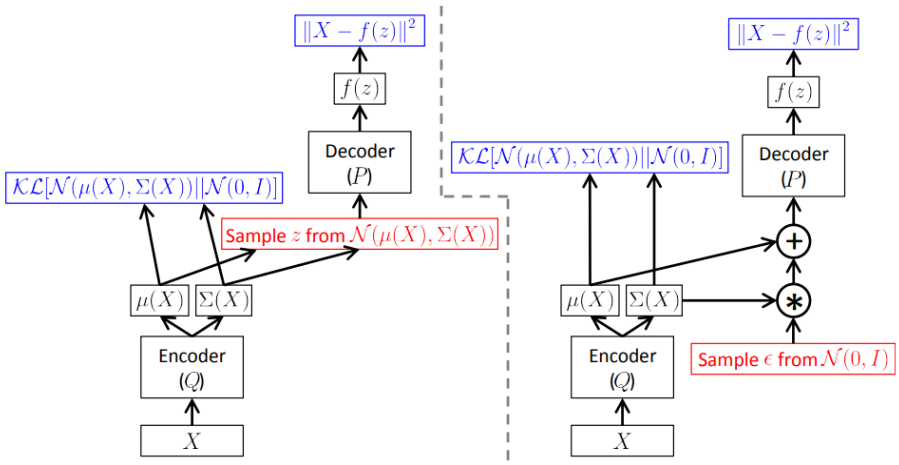

Figure 2: Diagram for Variational Autoencoders from [6].

In this part, we start with building a Autoencoder and then transform it into a VAE. Finally we conduct analysis on VAEs.

1. (5%) Build an autoencoder with the design guidelines. The encoder starts 4 channels and the decoder 64 channels.

   Train the autoencoder with L2 reconstruction loss using the training set of CUFS.

   Plot the loss over training iterations. Visualize a couple of reconstructions of the training sketches every $1,000$ iterations.

   Show several reconstructions of the testing sketches.

2. (20%) Build a VAE as shown in Fig. 2 Left.

   More specifically, the encoder outputs a 128-d vector, which contains 64-d of means and variances each. Sample z from Gaussian distributions of the outputted means and variances. Decode z using a decoder.

   To simplify the problem, normalize the input images so that the values are $\in [0, 1]$. Also, use Sigmoid activations in the output layer of the decoder.

   The two losses are 1) marginal likelihood as L2 loss and 2) KL divergence.

   Marginal likelihood can thus be computed as

$$\mathcal{L}_{ML} = -\frac{1}{n}\sum_{i}^{n}\big(X_i \log(f(z_i)) + (1 - X_i)\log(1 - f(z_i))\big),$$

where X is the input image, $f(z)$ is the reconstruction, and n is the batch size.

The other loss is KL divergence between $\mathcal{N}(\mu, \Sigma)$ and $\mathcal{N}(0, I)$, or

$$\mathcal{L}_{KL} = \frac{1}{2n} \sum_{i}^{n} (\mu_i^2 + \sigma_i^2 - \log(\sigma_i^2) - 1),$$

where μ and σ are the outputs from the encoder.

Train the VAE with the two loss weighted equally using the training set of CUFS.

Plot the loss over training iterations. Visualize a couple of reconstructions of the training sketches every $1,000$ iterations.

Show several reconstructions of the testing sketches and the ones from random sampled z.

3. (10%) In this question, you can either
   - Improve the VAE by adding regularization, adding noise, or any other methods.
   - Visualize and analyze the learned manifold. For example, compute z's from two images and show a series of reconstructions of the combinations of z's.
   - Train a larger VAE on the CelebA dataset.

   We might give extra credits if you finish multiple points.

2 Generative Adversarial Networks (35%)

Generative adversarial networks (GANs) [7] is a framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G.

In this part, you will first build a basic version of GANs that is suitable for modeling natural images, i.e., deep convolutional generative adversarial networks (DCGANs) [14]. After that, you will need to implement an improved GAN of your choice.

1. (10%) We encourage you to read [14] before starting this part.

   Build a generator G and a discriminator D by following the design guidelines. You may use 100-d z as a starting point.

The loss for the discriminator D is

$$\mathcal{L}_D = -\frac{1}{2n}\sum_i \left(\log D(X_i) + \log(1 - D(G(X_i)))\right),$$

where D and G denote the discriminator and generator, respectively.

The loss of the generator G is

$$\mathcal{L}_G = -\frac{1}{n}\sum_i \log D(G(X_i)).$$

You should train D twice and G once for one iteration. The learning rate for G should be higher than D by $5 - 10$ times.

Train the DCGAN on the CUFS training set. Plot the losses over training iterations. Visualize a couple of generated images of the training sketches every $1,000$ iterations.

Show several generated sketches from random sampled z. State your observation and explain the reason behind it.

2. (10%) Following the previous question, train a larger DCGAN on the CelebA dataset.

   Plot the losses over training iterations. Visualize a couple of generated images of the training sketches every $1,000$ iterations.

   Show several generated sketches from random sampled z. Check if the problem in the previous question still exists or not. Explain the reason.

3. (15%) In this question, you are asked to implement a variant of GANs and demonstrate it on the CUFS or CelebA dataset of your choice.

   Some candidates are as follows:

   - Conditional GAN [13] where additional inputs are added to both G and D. For example, you can use the attributes in CelebA dataset to control the attributes of generated images.
   - Improved GAN [15], where feature matching and minibatch discrimination are added to prevent the mode collapse problem.
   - Wasserstein GAN [4], where the losses are modeled approximately using Wasserstein metrics. The Wasserstein metrics provide a more directional guidance compared to KL divergence.

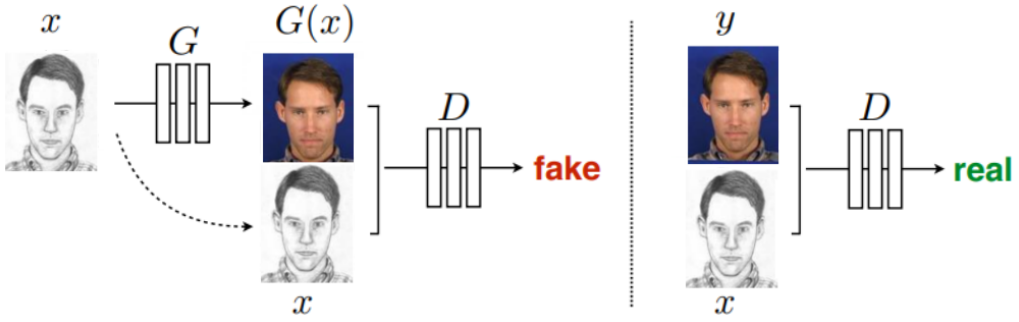

Figure 3: Diagram for illustrating pix2pix networks. The example shown here is generating photos given sketches.

- Energy-based GAN [16], where autoencoders are used to model the discriminator. The resulting loss functions are derived from energy.
- Or any other advanced GAN, such as LSGAN [12], WGAN-GP [8], BEGAN [5], etc.

Note that you do not need to implement every detail in the model so long as the major loss/regularization is correctly incorporated.

Plot the losses over training iterations. Visualize a couple of generated images of the training sketches every $1,000$ iterations. Show several generated sketches from random sampled z.

We might give extra credits if your implementation performs exceptionally well.

3 Image-to-Image Translation (30%)

In this part, you will implement image-to-image translation, or pix2pix [9]. We encourage to read the paper before starting this part.

You should download the CUFS dataset form the website [3]. They provide three sets of photo-sketch dataset, i.e., CUHK student data set, AR data set, and XM2GTS data set. Use any one of them for this part.

1. (20%) Build a pix2pix model as shown in Fig. 3.

   The generator G can be constructed by concatenating a decoder and an encoder. In the middle layer of G, add a dropout layer.

The loss of G consists of the original adversarial loss and L1 loss. We recommend to weight the L1 loss 10 times more than the adversarial loss as a starting point.

The discriminator D can be constructed using a decoder where the output resolution is 4×4. The loss of D is similar to the one in part 2 except that now it is a pixel-wise cross-entropy loss. (It is thus called PatchGAN.)

- Train the network with only G in the photo-to-sketch direction, i.e., input photos and output sketches.
- Train the network completely with D and G in the photo-to-sketch direction.
- Train the network completely with D and G in the sketch-to-photo direction.
- Construct G with skip connections (concatenating encoder and decoder corresponding layers' activations). Train the network completely with D and G in the photo-to-sketch direction.

For each bold point, plot the losses over training iterations and show several input-output pairs on the test set. Compare all the results.

(If you run out of time, complete the second and the third as you can proceed to the next question.)

2. (10%) Play with the trained model.
   - Take selfies of all team members' faces and generate sketches using the trained model.
   - Sketch all team members' faces and generate photos using the trained model.
   - Do anything creative [2] or play with other faces of your choice.

   Note that you should align the facial landmarks and match the background to best exploit the trained model.

References

[1] CelebA dataset. http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.

[2] Edges to cats. https://twitter.com/hashtag/edges2cats?src=hash.

[3] CUFS dataset. http://mmlab.ie.cuhk.edu.hk/archive/facesketch.html.

[4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[5] D. Berthelot, T. Schumm, and L. Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[6] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[8] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[9] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[10] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[11] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. *arXiv preprint ArXiv:1611.04076*, 2016.

[13] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[14] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[15] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[16] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.