CIS680: Vision & Learning
Assignment 3: Deep Generative Models
Due: Dec. 12, 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. You should finish at least any one part of this assignment no matter how big your final project is. The guideline is that most people should be fine only implementing one question, but if your project is smaller in scale, you will need more.
- There is no single answer to most problems in deep learning, therefore the questions will often be underspecified. You need to fill in the blanks and submit a solution that solves the (practical) problem. Document the choices (hyperparameters, features, neural network architectures, etc.) you made in the write-up.
- The assignment will describe the task on a high level. You are supposed to find out how to complete the assignment in the programming framework of your choice. While the text of the assignment should be sufficient to understand the task, you are welcome to read the references that will describe the used concepts in more detail.
- All the code should be written in Python. You should use either Tensorflow or PyTorch to complete this homework.
- You must submit your solutions online on **Canvas**. You should submit one folder for each part. Submit your code compressed into a single ZIP file named "<penn_key>.zip". Jupyter notebooks are acceptable. Submit your **PDF report** to a separate assignment called "HW3 PDF Submission". Note that you should include all results (answers, figures) in your report.

Overview

This homework aims at implementing, analyzing, and training deep generative models. Explicit and implicit generative models were traditionally important for unsupervised learning and representation learning. Recently, the demonstrated ability to generate very convincing high-resolution images [7, 13] led to excitement in various application areas, as graphics [18], generative design and even the emerging movement of neural art [3]. In this assignment, you will implement the two most commonly used classes of deep generative models, Variational Autoencoders (VAEs) and generative adversarial networks (GANs). Unlike previous assignments where architectures are specified, you have design or find the architectures for this assignment on your own.

This homework consists of three parts.

1. Implement and analyze Variational Autoencoders (VAEs) [14].
2. Implement and analyze generative adversairal networks (GANs) [9].
3. Implement and play with image-to-image translation (pix2pix) [12].

Datasets

We mainly use two datasets in this assignment: CUHK Face Sketch Database (CUFS) [4] and Large-scale CelebFaces Attributes (CelebA) [1] Datasets.

CUFS serves as a testbed where you can experiment with your architectures. You will evaluate your network on CelebA after you are sure that you know how it behaves.

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.

(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.

- 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.

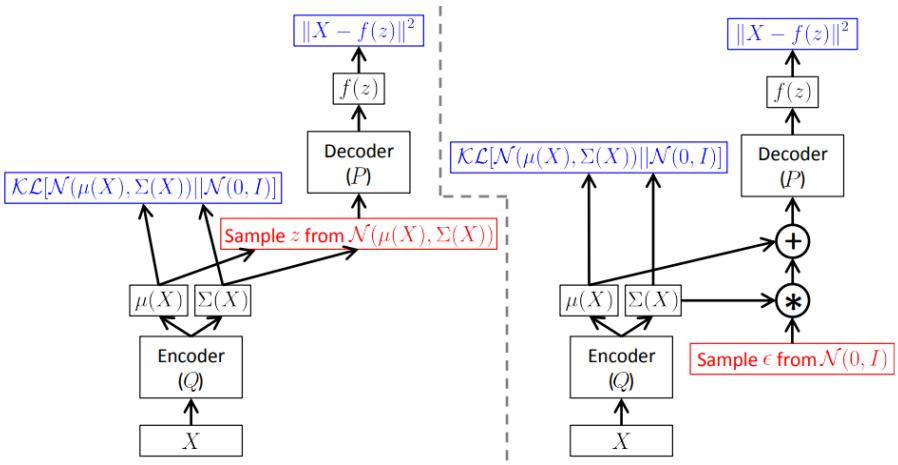

Figure 2: Diagram for Variational Autoencoders from [8].

1 Variational Autoencoders (35%)

Variational Autoencoder (VAE) is a latent variable model that allows to learn generative models of complex distributions (such as images) with high accuracy. We encourage you to read at least one of the two papers [14, 21]. You can also take a look at the more easily digestible tutorial by Carl Doertsch [8].

While VAE takes the form of an autoencoder, the two models have completely different objectives and use cases. Autoencoders [11] compress the data and are used for representation learning. VAEs learn the probability distribution of the data and can be used to generate novel samples from this distribution, something that is considerably more complicated with vanilla autoencoders.

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

1. (5%) Build an autoencoder with the design guidelines. You can start with latent dimensionality of 64 channels. Train it with L_2 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 output means and variances. Decode z using the decoder. To make the variances σ the network outputs positive, you should output $\log \sigma$ instead of σ from the encoder. You will likely have to modify the loss to work with $\log \sigma$ as exponent followed by logarithm can lead to numerical instability.

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

The two losses are 1) posterior likelihood and 2) KL divergence.

Marginal likelihood can 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 on the training set of CUFS. Note that depending on your setup, you might need to balance the losses with a hand-tuned constant.

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.

   Finishing multiple points warrants extra credits or can be used as a substitution for later questions.

2 Generative Adversarial Networks (35%)

Generative adversarial networks (GANs) [9] learn an implicit generative models of images via an adversarial process, in which two models are trained simultaneously. The generative model G produces images given noise, and the discriminative model D estimates the boundary between the samples from training data and generated by G. The network G tries to move the generated images closer to the boundary defined by D, which creates the adversarial game.

In this part, you will first build a basic version of GANs that is suitable for modeling natural images, deep convolutional generative adversarial networks (DCGANs) [20]. After that, you will need to implement an improved GAN of your choice.

1. (10%) We encourage you to read [20] 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

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

   Train the DCGAN on the CUFS training set. Plot the losses over training iterations. Visualize a couple of generated images 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 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 [16] 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.
   - Spectral Normalization [17] is a normalization method designed to constrain the lipschitz constant of the discriminator mapping.
   - Variational Discriminator Bottleneck [19] is a method of stabilizing the discriminator training by using the Information Bottleneck principle. While the theory of the method is somewhat involved, the implementation is relatively straightforward.
   - Improved GAN [22]. From here, you can take feature matching or minibatch discrimination, which are used prevent mode collapse.
   - Wasserstein GAN [5], which uses a loss based on the earth-mover distance.
   - Energy-based GAN [23], where autoencoders are used to model the discriminator.
   - Any other GAN variation, such as LSGAN [15], WGAN-GP [10], BEGAN [6], or anything else you can find.

   You only need to implement the main proposed technique (or one of the proposed techniques) of the paper.

   Plot the losses over training iterations. Visualize a couple of generated images 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 as per Isola et. al. [12]. We encourage you to read the paper before starting this part.

You should download the CUFS dataset form the website [4]. They provide three sets of photo-sketch dataset, i.e., CUHK student data set, AR data set, and XM2GTS data set.

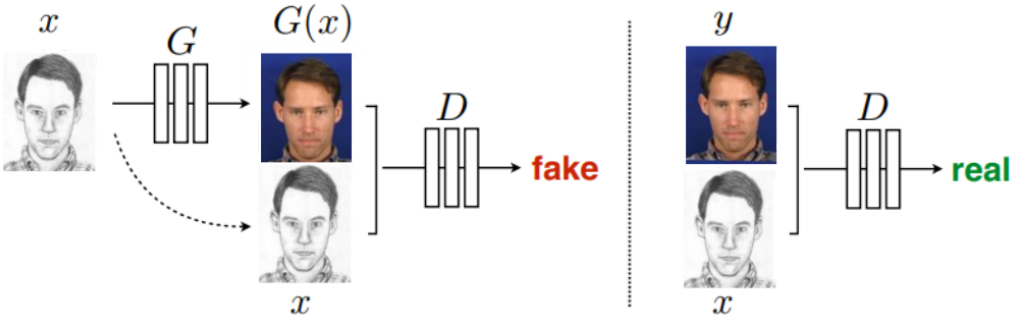

Figure 3: Diagram illustrating the pix2pix model. In the example, photos are generated given sketches.

Use any one of them for this part.

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

   The generator G architecture is roughly a decoder followed by 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 e.g. 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. Because of this, no part of the network observes the whole generated image. This type of discriminator is 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.
   - (Extra) 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.

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] Has artificial intelligence brought us the next great art movement? here are 9 pioneering artists who are exploring ais creative potential. `https://news.artnet.com/market/9-artists-artificial-intelligence-1384207`.

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

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

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

[7] A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

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

[9] 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.

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

[11] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *International Conference on Learning Representations*, 2018.

[18] K. Nagano, J. Seo, K. San, A. Hong, M. Goldwhite, J. Xing, S. Rastogi, J. Kuang, A. Agarwal, H. Kung, et al. Deep learning-based photoreal avatars for online virtual worlds in ios. In *ACM SIGGRAPH 2018 Real-Time Live!*, page 1. ACM, 2018.

[19] X. B. Peng, A. Kanazawa, S. Toyer, P. Abbeel, and S. Levine. Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow. *arXiv preprint arXiv:1810.00821*, 2018.

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

[21] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

[22] 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.

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