CIS 680: Advanced Machine Perception
Assignment 4: Generative Adversarial Network
Due: December 9, 2019 at 11:59pm

1 Overview

A generative model has huge difference from a discriminative models which you have encountered in earlier homework. The discriminative model is able to determine which class the given input belongs to but it cannot imagine any instance of a specific class. However, the generative model can generate such instances as it is named. In this exercise, you will implement a family of generative models including a variational autoencoder (VAE) and a generative adversarial network (GAN) [1].

2 Basics of Generative Models

2.1 Variational Auto-Encoders (VAEs)

Variational Auto-Encoders (VAEs) is a generative model built on top of auto-encoder. The encoder network encodes an input a compact Gaussian distribution, perturbs the latent encoding by adding a sampled noise, and decodes the latent code into the reconstructed input by decoder network.

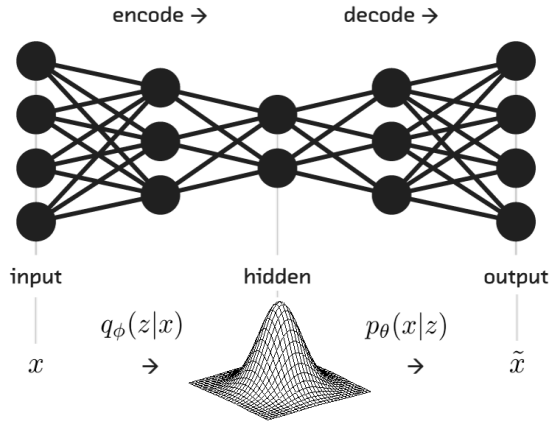


Figure 1: Variational Auto-Encoders (VAEs)

The loss function for training VAEs contain two parts. The first part is a pixel-wise reconstruction loss that enforces the reconstructed outputs are similar to the inputs. The second part is Kullback-Leibler (KL) divergence, which enforces the latent space to be a Gaussian distribution so that we can sample from during inference.

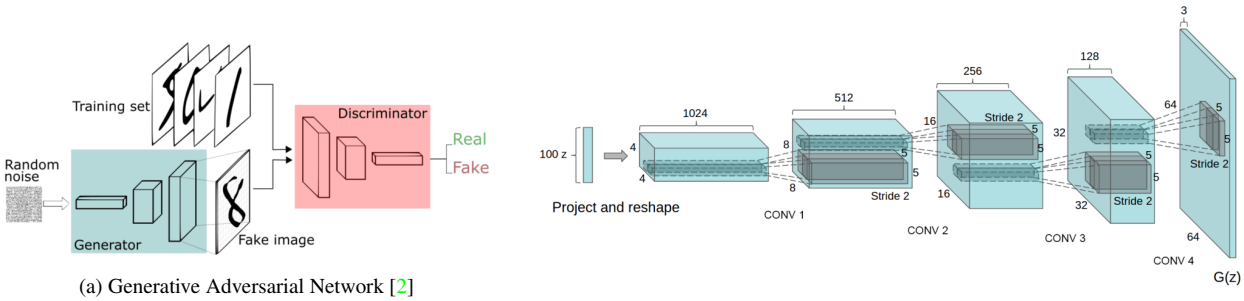
The VAE formulation is shown as follows,

$$\begin{aligned} &\log P(x) - KL[Q(z|x)||P(z|x)] \\ &= \mathbb{E}_{z\sim Q}[\log P(x|z)] - KL[Q(z|x)||P(z|x)] \end{aligned} \tag{1}$$

where $E_{z\sim Q}[\log P(x|z)]$ is the log-likelihood modeled by the decoder, and the KL term refers to Kullback-Leibler divergence between the posterior distribution $Q(z|x)$ modeled by the encoder and the prior distribution $P(z)$. In VAE, we assume $P(z)$ is a normal distribution $\mathcal{N}(0, 1)$ and the encoder estimates a mean $\mu(x)$ and standard deviation $\sigma(x)$ to approximate the posterior normal distribution Q, where $Q(z|x) = Q(z|\mu(x), \sigma(x))$.

Based on equation (1), the objective is to maximize the log-likelihood term and minimize the Kullback-Leibler divergence term. To maximize the log-likelihood, we directly apply the Euclidean distance loss or binary cross entropy.

2.2 Generative Adversarial Networks (GANs)



(a) Generative Adversarial Network [2]

(b) DCGAN [3]

Figure 2: Structure of GAN and DCGAN

A generative adversarial network (GAN) is a generative model composed of two neural networks: a generator and a discriminator. Contrary to VAE, these two networks are trained in unsupervised way via competition. The generator creates "realistic" fake images from random noise to fool the discriminator, while the discriminator evaluates the given image for authenticity (Fig 2a).

The loss function that the generator wants to minimize and the discriminator to maximize is as follows:

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2}$$

Here, G and D are the generator and the discriminator. The first and second term of the loss represent the correct prediction of the discriminator on the real images and on the fake images respectively.

2.3 Implementation Details

VAE

- You will implement fully-connected VAE model on the MNIST dataset which should be flattened into one-dimensional vector of 784 dimension.
- The details of VAE model is described in Table 1. Here, the encoder estimates the latent Gaussian mean and standard deviation using layer 2a and layer 2b. The decoder (layer 3 and 4) decodes the reparametrized code back into reconstructed input.
- You should study on your own to figure out how to implement KL divergence and reparametrization.
- The default weight between KL divergence and the reconstruction loss is 1:1 which can vary for better performance.

DCGAN

- You will implement deep convolutional GAN model on the STL-10 dataset which has built-in data reader of pytorch. You need proper transformation to normalize and resize image to 64×64.
- The details of the generator of DCGAN is described in Table 2. The discriminator should be literally opposite of the generator except leaky Relu and sigmoid instead of Relu and hyperbolic tangent respectively.
- You could start with batch size of 128, input noise of 100 dimension and Adam optimizer with learning rate of 2e-4. You may vary these hyperparameters for better performance.

Layer	Hyperparameters
FC 1	Hiddens=400, ReLU
FC 2a	Hiddens=20, ReLU
FC 2b	Hiddens=20, ReLU
FC 3	Hiddens=400, ReLU
FC 4	Hiddens=784, Sigmoid

Table 1: Network details of VAE

Layer	Hyperparameters
TransposedConv 1	Kernel = (4, 4, 1024), Padding=0, BatchNorm, ReLU
TransposedConv 2	Kernel = (4, 4, 512), Stride=2, Padding=1, BatchNorm, ReLU
TransposedConv 3	Kernel = (4, 4, 256), Stride=2, Padding=1, BatchNorm, ReLU
TransposedConv 4	Kernel = (4, 4, 128), Stride=2, Padding=1, BatchNorm, ReLU
TransposedConv 5	Kernel = (4, 4, 3), Stride=2, Padding=1, Tanh

Table 2: Network details of DCGAN

2.4 Report

- **Visualization of Training:** Please report loss curve during the training, and 36 generated images (in 6-by-6 grid as Fig 2 of GAN paper [1]) at at least 5 different training iterations.
- **Qualitative Evaluations:** Please report 36 generated images (and corresponding original images if there are) after training.

3 Conditional Image Synthesis wtih GANs

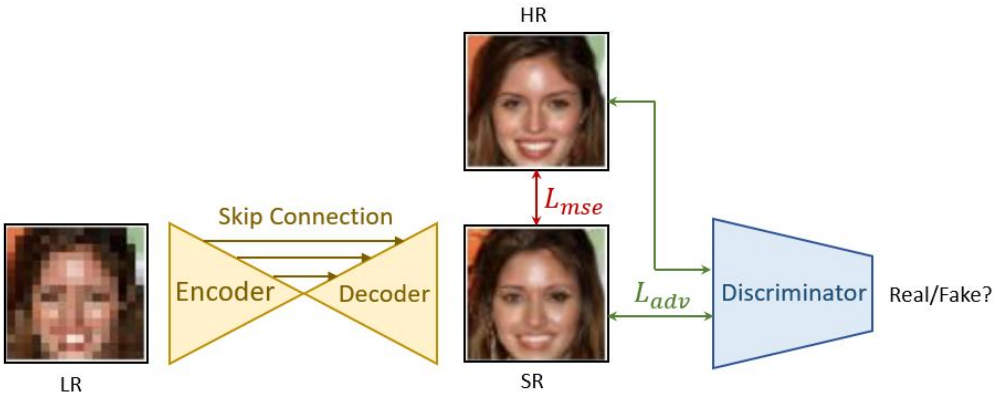


Figure 3: This is an overview of the super-resolution network. The generator is a Encoder-Decoder network with skip connections.

3.1 Implementation Details

In the super-resolution task, your network learns to upsample a 16x16 low-resolution image into a 64x64 image with refined visual details. There are total 20K training images and 1K testing image. The following are the implementation tips:

- The generator is an Encoder-Decoder network with skip connections. At every spatial resolution of the features, you need to concatenate features from encoder to the decoder. At the last layer of decoder, the feature is concatenated with the original input image.
- The low resolution image is obtained by downsampling the high-resolution into 16x16 and then upsampling back into 64x64 with bilinear interpolation. After this operation, the low-resolution will have the same dimension (64x64) as the high-resolution, but without any fine-level visual details.
- For the GAN loss, the default is to binary cross entropy loss, which is the same as Q1. However, you are encouraged to utilize any other improved GAN loss function, such as LSGAN, WGAN, WGAN-GP, Spectral Normalization GAN, for stablizing training or better quality.
- For the hyperpameters, you could use batch size of 32, 100 epochs, and Adam optimizer with 2e-4 learning rate. You may vary these hyperparameters for better performance.
- The network is jointly optimized by a pixel-wise reconstruction loss and GAN loss. In GAN updates, the generator G and discriminator D is updated alternatively, and the reconstruction can be added into part of the

Encoder	
Layer	Hyperparameters
Conv 1	Kernel = (4, 4, 32), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 2	Kernel = (4, 4, 64), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 3	Kernel = (4, 4, 128), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 4	Kernel = (4, 4, 256), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 5	Kernel = (4, 4, 512), Stride=2, Padding=1, BatchNorm, LeakyReLU

Decoder	
Layer	Hyperparameters
TransposedConv 1	Kernel = (4, 4, 256), Padding=0, BatchNorm, LeakyReLU
TransposedConv 2	Kernel = (4, 4, 128), Padding=0, BatchNorm, LeakyReLU
TransposedConv 3	Kernel = (4, 4, 64), Padding=0, BatchNorm, LeakyReLU
TransposedConv 4	Kernel = (4, 4, 32), Padding=0, BatchNorm, LeakyReLU
TransposedConv 5	Kernel = (4, 4, 16), Padding=0, BatchNorm, LeakyReLU
TransposedConv 6	Kernel = (3, 3, 3), Padding=1, Tanh

Discriminator	
Layer	Hyperparameters
Conv 1	Kernel = (4, 4, 32), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 2	Kernel = (4, 4, 64), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 3	Kernel = (4, 4, 128), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 4	Kernel = (4, 4, 256), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 5	Kernel = (4, 4, 512), Stride=2, Padding=1, BatchNorm, LeakyReLU
Conv 6	Kernel = (3, 3, 1), Stride=2, Padding=1, Sigmoid

loss for updating G. The default weights between recon loss and GAN loss is 1:1. Again, please feel free to adjust the weights as you like for better performance.

3.2 Evaluation and Report

- **Visualization of Training:** For a fixed input image, please show five intermediate results at different training iteration to show how the quality gets improved over time. You can choose the input and evaluation iterations.
- **Quantitative Evaluations:** To evaluate the quality of the generated images, you will need to use two types of evaluation metrics. The first is the pixel-wise MSE distance between the generated and ground-truth images. The second is a perceptual distance LPIPS computed from CNN feature space. You should compute the scores under both evaluation metrics and average the results of all 1,000 testing images.
  Please refer LPIPS github for computing this LPIPS score: https://github.com/richzhang/PerceptualSimilarity
- **Qualitative Evaluations:** Please provide 10 images with the corresponding low-resolution input, generated output, and ground-truth high-resolution. You can put the input, generated and ground-truth outputs in a row for comparison.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems* 26, pages 2672–2680, 2014. 1, 3

[2] Thalles Silva. An Intuitive Introduction to Generative Adversarial Networks (GANs). FreeCodeCamp.org, Jan. 2018, www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/. 2

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *Proceedings of 4th International Conference on Learning Representations*, 2016. 2

[4] Mehdi Mirza, and Simon Osindero. Conditional Generative Adversarial Nets. arXiv preprint arXiv:1411.1784, 2014.