Let's Play GANs with Flows and friends

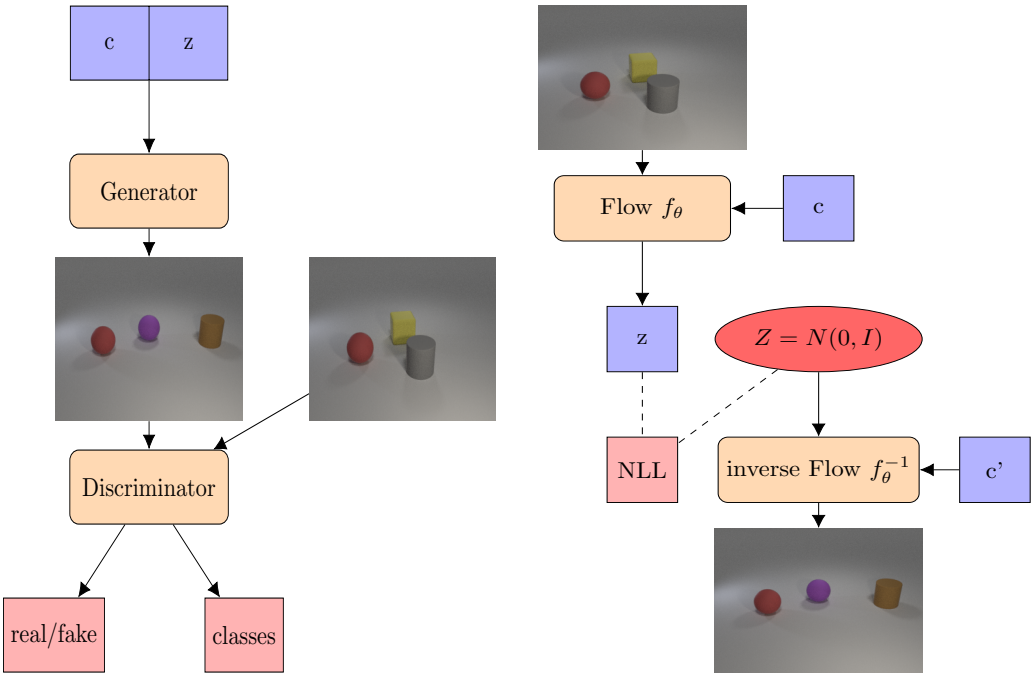
Peng-Yu, Chen

- Deadline: June 15, 2021 11:59 a.m.
- Format: Experimental report (.pdf) and source code (.py). Zip all files in one file and name it like DLP_LAB7_yourstudentID_name.zip. e.g. DLP_LAB7_309551113_Brian.zip.

1 Lab Description

In this lab, you need to implement generative adversarial network (GAN) and a normalizing flow(NF) network for 2 tasks.

For the first task, both of your generative models should be conditional. They should be able to generate synthetic object images with multi-label conditions. For example, given "red cube" and "blue cylinder", your model should generate the synthetic images with red cube and blue cylinder (see in Fig. 1 for an example). After generation, you need to input generated images to a pre-trained classifier for evaluation.



(a) Illustration of training procedure using cGAN. (b) Illustration of training procedure using cNF.

Figure 1: Illustration of training procedure for object images generation

The second task is human face generation. You should train your normalizing flow model to generating human faces. You should make use of it to accomplish several applications, which are explained in detailed at next section.

Note that some applications in second task can be done without conditioning.

2 Requirements

- **Task 1: Object images generation**
  - Implement a conditional GAN
    - Choose your cGAN architecture
    - Design your generator and discriminator
    - Choose your loss function
    - Implement the training function, testing function, and data loader
  - Implement a conditional NF
    - Choose your conditional NF architecture
    - Choose your flow model
    - Implement the training function, testing function
  - Generate the synthetic images
    - Evaluate the accuracy of test.json & new_test.json
- **Task 2: Human faces generation**
  - Implement a NF model
    - (Optional) Choose your conditional NF architecture
    - Choose your flow model
    - Implement the training function
    - Finish applications:
      - Conditional face generation: Generate faces given attributes as conditions. You should not use latent vector that mapped to any input image for generation, and conditions should be set by yourself
      - Linear interpolation: Pick 2 images, you should interpolate at least 5 images according to their latent representations. See Fig. 2

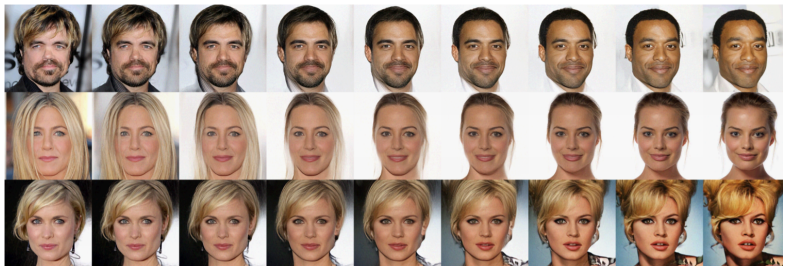


Figure 2: Illustration of linear interpolation [8]

      - Attribute manipulation: Given an attribute, let's say "smiling", you should calculate mean vector z_{pos} of images with "smiling" and z_{neg} of images without "smiling". Then you choose 1 image, and modify its attribute using vector $z_{pos} - z_{neg}$ with different scaling. See Fig. 3

Figure 3: Attribute manipulation of "smiling". From negative to positive [8]

3 Implementation Details

3.1 GAN

3.1.1 Architecture of conditional GAN

There are variants of conditional GAN architectures. You can choose one of the models in the following list or a hybrid version as your conditional GAN structure

- conditional GAN [13]
- Auxiliary GAN [15]
- Projection discriminator [14]

3.1.2 Design of generator and discriminator.

Many GAN structures are developed to generator high quality images. Basically, all GAN architecture are based on DCGAN which is mainly composed of convolution neural network. You can adopt one of the structures or a hybrid

- DCGAN
- Super resolution GAN [9]
- Self-attention GAN [16]
- Progressive growing of GAN [7]

3.1.3 Training loss functions

Besides the significant progress of GAN architectures, the training function also plays an important role. Your loss function should combine with your choice of cGAN as it will take a part in your training function, e.g., auxiliary loss. Here are the reference training loss functions.

- GAN [4]
- LSGAN [12]
- WGAN [2]
- WGAN-GP [5]

3.2 Normalizing Flow

3.2.1 Conditioning on NF model

Again, there's also several different architecture of conditional NF models. You can pick one in list below.

- cINN [1]
- SRFlow [11]
- c-Glow [10]

3.2.2 Architecture of NF model

Normalizing flow models have shown their competitive performances as a generative model. Here's some models that are useful in practice. Again, feel free if you want to adopt other methods that are not listed.

- Real NVP [3]
- Glow [8]
- Flow++ [6]

3.3 Other implementation details

- Your can ignore previous suggestions and choose any GAN & NF architecture you like.
- For task 1:
  - Except for the provided files, you cannot use other training data (e.g. background image).
  - Use the function of a pre-trained classifier, *eval*(images, labels) to compute accuracy of your synthetic images.
  - The same object will not appear twice in an image.
  - The normalization of input images is transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)). You should not apply normalization on your generated images but apply de-normalization while generating RGB images.
- For task 2:
  - You can finish the second & third applications without conditioning.
- Make use of *make_grid*(images) and *save_image*(images, path) from torchvision_utils import save_image, make_grid to save your image (8 images a row, 4 rows).
- Here's some tips that you may find useful when training GAN `https://github.com/soumith/ganhacks`.

4 Dataset Descriptions

You can download the dataset.zip file from google drive. The link is in dataset.txt on new E3. You will get 2 folders: task_1 & task_2

- task_1:
  - There will be 6 files, classifier_weight.pth and a folder "images". The 6 files are: readme.txt, train.json, test.json, object.json, dataset.py and evaluator.py. All the details of the dataset are in the readme.txt.
  - One additional file *new_test.json* will be released 5 days before deadline for scoring.
- task_2: There will be 1 folder and 3 files: CelebA-HQ-img with all images in it, dataset.py, CelebA-HQ-attribute-anno.txt and README.txt. All the details of the dataset are in the README.txt.

5 Scoring Criteria

1. Report
   - Introduction (10%)
   - Implementation details (15%)
     - Describe how you implement your model, including your choices in Section 3.1 & 3.2. (10%)
     - Specify the hyperparameters (learning rate, epochs, etc.). (5%)
   - Results and discussion (75%)
     - Task 1 (45%)
       * Show your results based on the testing data. (including images) (5%)
       * Classification accuracy on test.json and new_test.json using cGAN and cNF. (5% on test.json, 10% on new_test.json for each model)
       * Discuss the results of different models architectures. (10%)
         For example, what kinds of condition design is more effective to help cGAN, or the advantages of using NFs as generative models.

Accuracy	Grade
score ≥ 0.8	100%
$0.8 >$ score ≥ 0.7	90%
$0.7 >$ score ≥ 0.6	80%
$0.6 >$ score ≥ 0.5	70%
$0.5 >$ score ≥ 0.4	60%
score < 0.4	0%

- Task 2 (30%)
  - Show your results & explain implementation details for following applications (10% each):
    - Conditional face generation: At least 4 images with at least 3 conditions.
    - Linear interpolation: 3 pairs of images with at least 5-image interpolations.
    - Attribute manipulation: At least 2 attributes of same image.
  - Note that you will get 0 if there's no explanations for one application result
  - Quality of results will be judged subjectively by TAs.

References

[1] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks, 2019.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp, 2017.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[6] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design, 2019.

[7] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2017.

[8] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions, 2018.

[9] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016.

[10] You Lu and Bert Huang. Structured output learning with conditional generative flows, 2020.

[11] Andreas Lugmayr, Martin Danelljan, Luc Van Gool, and Radu Timofte. Srflow: Learning the super-resolution space with normalizing flow, 2020.

[12] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2016.

[13] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[14] Takeru Miyato and Masanori Koyama. cgans with projection discriminator, 2018.

[15] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans, 2016.

[16] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2018.