

EESM5900V

Assignment 3

Introduction

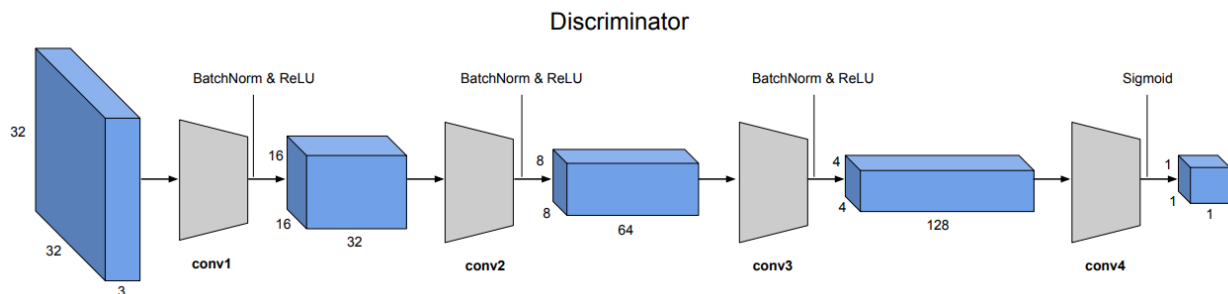
In this assignment, you'll get hands-on experience coding and training GANs. We will implement a specific type of GAN designed to process images, called a Deep Convolutional GAN (DCGAN). We'll train the DCGAN to generate emojis from samples of random noise.

We provide the skeleton code written with Pytorch for your convenience. Note that the skeleton code contains some implementation about CycleGAN, which you can ignore.

Part 1: Deep Convolutional GAN (DCGAN)[60%]

For the first part of this assignment, we will implement a Deep Convolutional GAN (DCGAN). A DCGAN is simply a GAN that uses a convolutional neural network as the discriminator and a network composed of transposed convolutions as the generator. To implement the DCGAN, we need to specify three things: 1) the generator, 2) the discriminator, and 3) the training procedure. We will develop each of these three components in the following subsections.

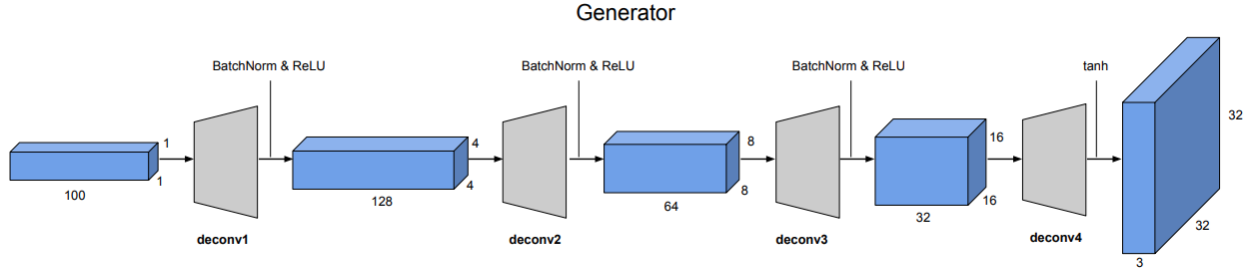
Implement the Discriminator of the DCGAN [20%]



Implement this architecture by filling in the `__init__` method of the `DCDiscriminator` class in `models.py`. Note that the forward pass of `DCDiscriminator` is already provided for you.

Implement the Generator of the DCGAN [20%]

Implement this architecture by filling in the `__init__` method of the `DCGenerator` class in `models.py`. Note that the forward pass of `DCGenerator` is already provided for you.



Implement the Training Loop [20%]

Next, you will implement the training loop for the DCGAN. A DCGAN is simply a GAN with a specific type of generator and discriminator; thus, we train it in exactly the same way as a standard GAN. The pseudo-code for the training procedure is shown below. The actual implementation is simpler than it may seem from the pseudo-code: this will give you practice in translating math to code.

Open up the file `vanilla_gan.py` and fill in the indicated parts of the train function following the pseudo-code shown below. The provided skeleton code basically follows the DCGAN [2] but used least-squares loss proposed in LSGAN [1].

Algorithm 1 GAN Training Loop Pseudocode

- 1: **procedure** TRAINGAN
- 2: Draw m training examples $\{x^{(1)}, \dots, x^{(m)}\}$ from the data distribution p_{data}
- 3: **Draw m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from the noise distribution p_z**
- 4: **Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \dots, m\}$**
- 5: **Compute the (least-squares) discriminator loss:**

$$J^{(D)} = \frac{1}{2m} \sum_{i=1}^m \left[\left(D(x^{(i)}) - 1 \right)^2 \right] + \frac{1}{2m} \sum_{i=1}^m \left[\left(D(G(z^{(i)})) \right)^2 \right]$$

- 6: Update the parameters of the discriminator
- 7: **Draw m new noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from the noise distribution p_z**
- 8: **Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \dots, m\}$**
- 9: **Compute the (least-squares) generator loss:**

$$J^{(G)} = \frac{1}{m} \sum_{i=1}^m \left[\left(D(G(z^{(i)})) - 1 \right)^2 \right]$$

- 10: Update the parameters of the generator
-

Report [40%]

You should run experiments with completed codes of DCGAN. For DCGAN, you should run at least 30 epochs. If you have powerful gpus, then you can run extra steps. You are required to

report following contents:

- Report the training loss graph of DCGAN.
- Report the generated image samples of DCGAN.

Submission

You should submit your assignment in the format of a zip file. The name of a submission file should be like 'PA3-{your name}-{your student id}.zip'. You must contain these files in your submission.

- models.py
- vanilla_gan.py
- report.pdf

References

- [1] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.
- [2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.