CSA 250: Deep Learning
(Spring 2020)

Sargur N. Srihari

Indian Institute of Science,
Bengaluru, Karnataka
email : srihari@buffalo.edu

March 22, 2020

1 Introduction

This project is about learning a generative model from which samples can be generated. In this project you will implement two types of Generative Adversarial Networks (GANs): One known as the Deep Convolution GAN (DCGAN); the other known as Self-Attention GAN (SA-GAN) using CIFAR-10 dataset. Basic DCGAN consists of two neural networks: a discriminator (D) and a generator (G) which compete with each other making each other stronger at the same time. One of the simple variants of GAN in Deep Convolutional GANs (DCGANs) in which G and D are both based on deep convolution neural network.

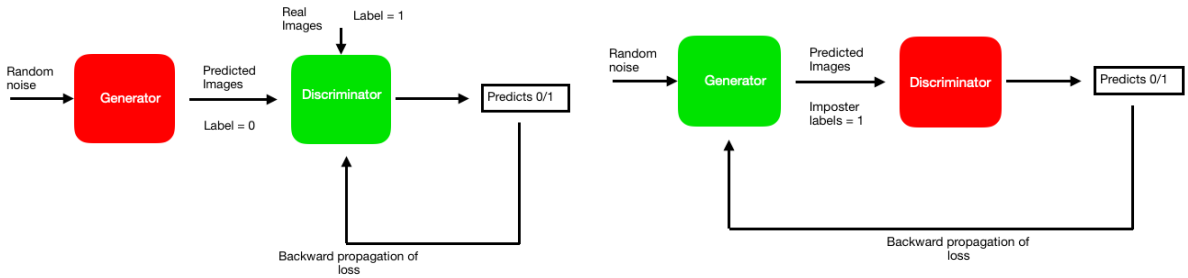

Figure 1: Training a basic Generative Adversarial Network. Figure on the left and right shows the training of a Generator and Discriminator respectively.

DCGANs exhibit limitations while modelling long term dependencies for image generation tasks. For example, dogs are often drawn with realistic fur texture but without clearly defined separate feet. The **problem with DCGANs** exists because model relies heavily on convolution to model the dependencies across different image regions. Since **convolution operator has a local receptive field**, long ranged dependencies can only be processed after passing through several convolutional layers. This could **prevent** learning about **long-term dependencies** for a variety of reasons:

1. A small model may not be able to represent them

2. Increasing the size of the convolution kernels can increase the representational capacity of the network but doing so also loses the computational and statistical efficiency obtained by using local convolutional structure

3. Mode collapse

To mitigate first two issues **self-attention** module is introduced in DCGANs. SA, exhibits a better balance between the ability to model long-range dependencies and the computational and statistical efficiency.

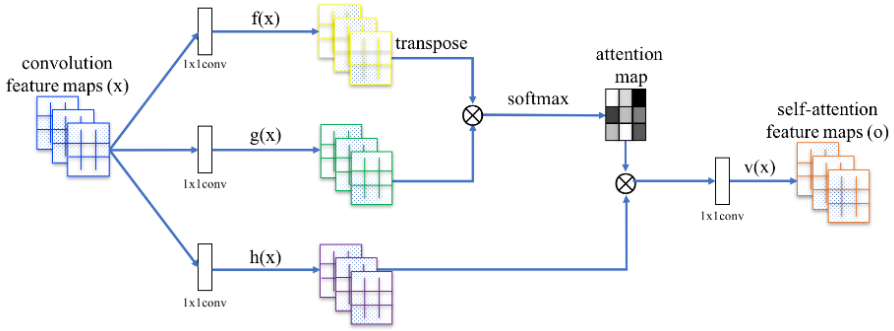

Figure 2: Architecture of a Self Attention module. The X denotes matrix multiplication. The softmax operation is performed on each row. This module is stacked after a selected convolution block present in a G and D.

To alleviate the third issue, GANs are trained with Wasserstein Loss. The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution q to the data distribution p. Even when two distributions are located in lower dimensional manifolds without overlaps, Wasserstein distance can still provide a meaningful and smooth representation of the distance in-between. The Equation for Wasserstein distance is given as follows:

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))] \tag{1}$$

In this project, students will use Wasserstein Loss while implementing a Self-Attention Generative Adversarial Network (SAGAN).

2 Project Requirements

2.1 GAN Model, Attention Implementation, Evaluation Metrics

1. Implement a model combining a conditional Generator (G) and Discriminator (D) using your own code.

2. Apply Spectral Normalization (SN) on G and D. Existing implementations for SN can be used.

3. Implement self-attention (SA) module using your own code.

4. Implement Wasserstein Loss function for the training using your own code.
5. Apply FID as an evaluation metric. Existing implementations for FID can be used.
6. It is expected that students validate the training epochs using Fréchet Inception Distance (FID) with a validation set. Existing implementations can be used
7. Compute the FID score during training at least at every 1000 iterations and plot it
8. A jupyter notebook containing the complete code.
9. The jupyter notebook should have an ultimate cell which when run executes and generates
   (a) 8 by 8 image grid
   (b) FID score. **CIFAR10 test data** should be used wile calculating FID

   form the best generator model obtained while training.

Optional Step: It is recommended to use the two-timescale learning rates update rule (TTUR) implemented in the SA-GANs paper in order for the GAN to converge to a local Nash equilibrium.
Guidelines for Using 3rd Party Code: In general third party code can only be used when imported from the deep learning API of the student's choice. Any other outside code used must only be "snippets" and must be cited. Students cannot share code between groups under any circumstances.

2.2 Report

2.2.1 Format

The project report must be written preferably in LaTeX.

2.2.2 Model and Training Description

Describe the implementation of the SAGAN model (including the attention component) and the training procedure. Describe what SAGAN is and how it differs from a DCGAN; explain what spectral normalization is; what attention is and why it is beneficial to many machine learning models. A results section should be included that summarizes the FID scores achieved while training DCGAN and SAGAN along with the training and validation loss curves.

2.2.3 Training Curves

Plots must be included in the report which shows the performance of the given model in terms of FID over the trained epochs.

2.2.4 Grid of Generated Images

A grid of images generated by the implemented SAGAN should be included. For some subset of classes in the dataset, both examples of images generated by the DCGAN and SAGAN should be included.

2.2.5 Table of Metrics for Performance of Different Models

A table of results including the mean FID for attention at level 32×32 should be included.

2.3 Extra Credit – Visualization of Attention Maps

Bonus points will be awarded for visualization of the attention layers.

3 Deliverables

The following documents/codes must be submitted:

1. Prepare a report and name it **Deep Learning Report 4.pdf**. In your report, you need to briefly describe what you have done, present the results (in a form you think is good) and provide a brief discussion of the results you obtained. Please feel free to play with the model architecture and hyper-parameter settings. Please make it concise and to-the-point.
2. You need to save the trained model in a respective format and put it in model directory.
3. Python code for training.
4. Generate 10 images from both of the models. Put the images generated by DCGAN in DCGAN directory and those generated by SA-GAN in SAGAN directory

The submission instructions will be provided in a separate document at a later stage. Please check piazza periodically for updates.