CPSC 8810: Deep Learning – Homework 3

by - Abhijeet Amle

***Abstract* — This paper explains the comparison between DCGAN and WGAN implemented on CIFAR-10 dataset.**

I. Introduction

GAN (Generative Adversarial Network) are algorithmic architectures that use two networks pitting against each other to produce synthetic new instances of data that will be similar like the real data. The generative network creates new data while the discriminative network will evaluate the data. Here, the generative network adapts to map from the latent space to the data distribution of interest, on the other hand, the discriminative network differentiates the data produced by the generative network from the true data distribution. The goal of the generative network is to generate novel data such that the discriminative network cannot determine between the data generated by the generative network and the true data of the dataset.

II. Problem Statement

Train a discriminator/generator pair on CIFAR-10 dataset utilizing techniques from DCGAN and Wasserstein GAN. Also, explain the difference between DCGAN and WGAN.

1. *Dataset: CIFAR-10*

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images.

The classes in the CIFAR-10 dataset are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

2. *DCGAN: Deep Convolutional Generative Adversarial Network*

DCGANs composes of mainly convolution layers without max pooling and fully connected layers. It uses transposed convolution and convolutional stride for up-sampling and down-sampling of the images. As DCGANs are hard to train, so there are architecture guidelines for stable training of the network.

- Use convolutional stride instead of max pooling
- For up-sampling use transposed convolution
- Eliminate fully connected layers.
- Use Batch normalization for all the layers except the input layer of the discriminator and the output layer for the generator.
- Use Leaky-ReLU in the generator except for the output layer where tanh activation function is used.
- Use Leaky-ReLU in the discriminator except for the output layer where sigmoid activation function is used.

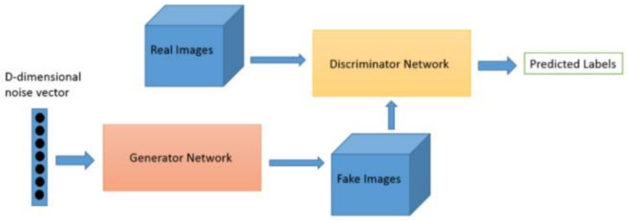
3. *WGAN: Wasserstein Generative Adversarial Network*

WGAN is an alternate way of training the generator model for better estimation of the data observed in the given training dataset. Usually the discriminator is used to predict or classify the probabilities from the generated images as being fake or real, but in the WGAN, the discriminator model is replaced with a critic that scores the fakeness or the realness of the image. The motivation behind WGAN is that training the generator should seek a minimization of the distance between the distribution observed in the generated examples and the distribution of the data observed in the training dataset. This distance is called as Wasserstein distance.

WGANs are more stable and are less prone to hyperparameters and model architecture.

Architecture guidelines for WGAN:

- In the output layer, instead of sigmoid activation function, use a linear activation function as the output is no longer a probability
- To train the generator model and the critic, use Wasserstein loss
- Instead of Adam, use RMSProp
- Use weight clipping
- Use a small learning rate and no momentum



(GAN architecture)

III. Implementation Details

1. *DCGAN*

For each epoch, trained the discriminator 5 times whereas trained the generator only once.

a) Generator:

- Used weights initializer as truncated normal initializer with standard deviation as 0.02
- Used transposed convolution layers with strides of 2
- Used batch normalization after each convolution layer with momentum=0.9 and epsilon=1e-5
- Used leaky ReLU as activation function after each layer
- Applied tanh activation function on the output of generator

b) Discriminator:

- Used weights initializer as truncated normal initializer with standard deviation as 0.02
- Used conv2d layers with strides of 2
- Used batch normalization after each convolution (except first) layer with momentum=0.9 and epsilon=1e-5
- Applied leaky ReLU on all the layer except last layer
- Applied sigmoid activation function to last layer of discriminator

c) Hyperparameters:

- Loss = reduce mean
- Optimizer = Adam with beta1=0.5 and beta2=0.9
- Learning rate = 2e-4
- Training batch size = 128

2. WGAN

For each epoch, trained the discriminator 5 times whereas trained the generator only once. Used weight clipping for the discriminator (applied on discriminator from variable scope).

a) Generator:

- Used weights initializer as truncated normal initializer with standard deviation as 0.02
- Used transposed convolution layers with strides of 2
- Used batch normalization after each convolution layer with momentum=0.9 and epsilon=1e-5
- Used leaky ReLU as activation function after each layer
- Applied tanh activation function on the output of generator

b) Discriminator:

- Used weights initializer as truncated normal initializer with standard deviation as 0.02
- Used conv2d layers with strides of 2
- Used batch normalization after each convolution (except first) layer with momentum=0.9 and epsilon=1e-5
- Applied leaky ReLU on all the layer except last layer

c) Hyperparameters:

- Wasserstein loss = mean of predicted and true
- Optimizer = RMSProp
- Learning rate = 2e-4
- Training batch size = 128

References

[1] https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/

[2] https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

[3] https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/

[4] https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/dcgan.ipynb

[5] https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/