Comparative study of DCGAN and WGAN on CIFAR10

*using TensorFlow in python

By- Abhinav

Abstract*— This work is to learn and understand how the Generative Adversarial Networks work and what is the mechanism behind the two most usable GAN, DCGAN and WGAN. The implementation will give a clear picture of how the discriminator and generator works in both of these models and which comes out with a better result.** **Keywords—** ***GAN, DCGAN, WGAN, critics, generator, discriminator etc.

I. Introduction

Generative Adversarial Network (GAN), is a technique which teaches two models simultaneously, a generator and a discriminator. As the name suggests a generator is a model that generates a new image and a discriminator is a model that tells the difference between the real and the fake.

In this work, I have used two GAN models, with slightly different approach and having their own advantages over the other. The following sections will cover the description of GAN and its two types which are DCGAN and WGAN. Further, there will the algorithm differences between both the WAN types and then at last will be the result of generating 10 images from each one of them.

II. Problem Description

Dataset:

For this project we have been using the dataset CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into 5 training batch and 1 test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. Classes are mutually exclusive.

Each batch files contains a dictionary with the following elements:

- data: 10000x3072 array of data. With the first 1024 containing the red channel values, the next 1024 containing the green channel value and the final 1024 the blue channel value.
- labels: 10000 numbers in the range 0-9. The number of index *i* indicates the label of the i^{th} image in the array data.
- label_names: a 10-element list which gives meaningful names to the numeric labels in the labels array described above.

Generative Adversarial Network (GAN):

GAN is a machine learning technique that teaches the two model simultaneously, one (Generator) taught to generate fake data and two (Discriminator) trained to distinguish between the real and the fake data. It is like a competition between the two models which is better and while doing that they keep on improving the others.

Generator generates the new data from a random noise, the objective is to create a new data having the characteristics of the training data, so that the new data looks indistinguishable from the real one. Discriminator on the other hand have to train itself in distinguishing between the newly generated data and the real one. If it finds the difference and declares the data generated by the generator as fake. This could be more understandable with an example of Mona Lisa painting originally painted by Leonardo da Vinci, someone counterfeits the painting to fool it with the original one but there is a counterfeit expert who can distinguish between the original and the fake one. Now, for the counterfeiter it is important to fool the expert by improving his counterfeiting techniques and for the expert it is important to be the best in finding the difference between the counterfeit and the original. In the same way a generator and a discriminator works. Generator improves itself by getting the feedbacks from the discriminator, if it gets the feedback as fake then it understands that it needs to improve and when it gets the feedback as real it knows that it did well in generating the data. The same way discriminator also learns, it learns how far is its prediction from the true labels. So, a generator gets better at producing realistic-looking data, the discriminator gets better at telling fake data from the real, and both the network continues to improve simultaneously.

There are two types of GAN on which I have worked on in this project, DCGAN and WGAN.

Deep Convolution Generative Adversarial Network (DCGAN):

The primary idea of DCGAN compared to the general GAN such as the multi-layer perceptron again is that it adds up sampling convolution layers between the input vector Z and output image in generator. In addition, in the discriminator it uses convolution layers like a regular convolution neural network to classify the generated and real images as the corresponding label real effect. Another idea is batch normalization, it takes a vector features or matrix and normalizes them so that they have the same parameter mean and

another parameter for the standard deviation so just like a standard multivariate Gaussian is how the features in the layers of a neural network are distributed. Another idea is they build on the ReLU activation with the leaky re which has a little negative slope.
Architectural guidelines:

- Replace any pooling layers with the strided convolutions (discriminator) and fractional-strided convolutions (generator)
- Use batchnorm in both the generator and the discriminator
- Remove fully connected hidden layers for deeper architectures
- Use ReLU activation generator for all layers except for the output, which uses Tanh
- Use LeakyReLU activation in the discriminator for all layers

Wasserstein Generative Adversarial Network (WGAN):
Instead of using a discriminator to classify or predict the probability of generated images as real or fake, the WGAN changes or replaces the discriminator model with a critic that scores the realness or fakeness of a given image.
The motivation behind this is to get rid of the problems of JS divergence by using Wasserstein Distance between the probability distribution of the actual distribution that we want to approximate and generated distribution. The tuning of parameters in generated image helps in getting closer to the distribution of actual distribution, and our objective is to minimize this difference between the two. The reason is that even when the two distribution are located in lower dimensional manifolds without overlaps, Wasserstein distance can still provide a meaningful and smooth representation of the distance in-between.
Architectural guidelines:

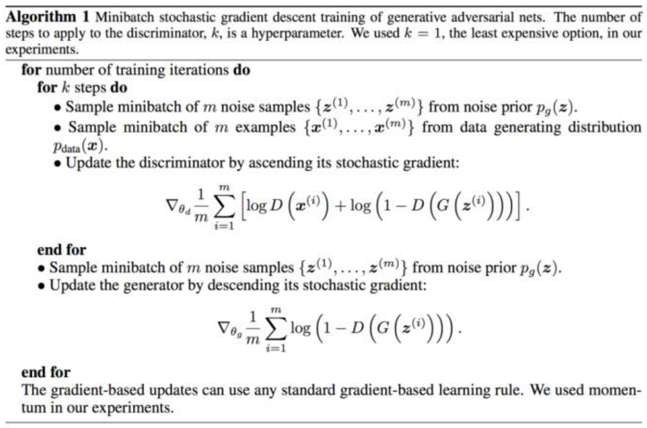
- No sigmoid as the output is no longer a probability
- Clipping the weight
- Training discriminator more than generator
- Using RMSProp instead of Adam
- Lower learning rate

Problem:
The problem is to generate the CIFAR images using the generator that can be closest to the original CIFAR data. Using the two different GAN model to understand the difference between the performance of the two.

III. APPROACH

Algorithm DCGAN:
The goal of the algorithm is to train the generator such that it produces result that could fool discriminator. The output from generator could be used as input for the discriminator. The output is the probability of the output closer to the original data when it is real and closer to the generated data given by 1-D(x). So the generator objective is to maximize D(z) or equivalently minimize 1-D(z). DCGAN have few architectural changes compared to GAN. The differences are as follows:

- Using strided convolution instead of pooling layers
- Using batch normalization in both generator and discriminator in each layers
- Using ReLU activation function



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$
    end for
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$
end for
```

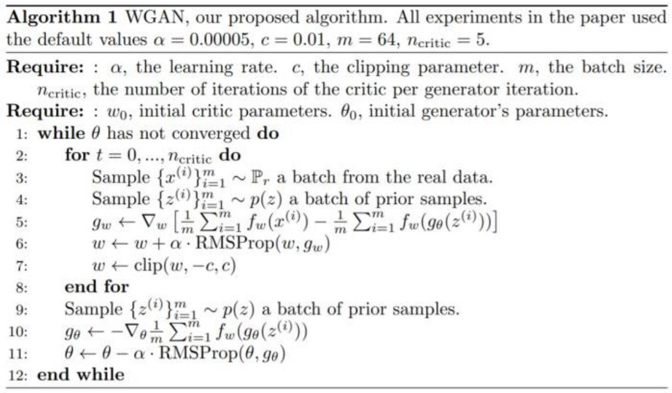
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Fig 1. GAN Algorithm

Algorithm WGAN:
There are few changes that is required for WGAN compared to DCGAN. The differences are as follows:

- No sigmoid function should be used in the output layer
- Using -1 for fake and 1 for real image instead of 0 and 1 respectively
- Using the Wasserstein loss to train the models
- Constraining the weights to a limited range after each mini batch update (e.g. [-0.01,0.01])
- Updating critic model more than generator, for this project it is 5
- Using RMSProp instead of Adam Optimizer

Below is the algorithm to train the WGAN model:



Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
 1: while $\theta$ has not converged do
 2:     for $t = 0, ..., n_{\text{critic}}$ do
 3:         Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
 4:         Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
 5:         $g_w \leftarrow \nabla_w \left[\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
 6:         $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
 7:         $w \leftarrow \text{clip}(w, -c, c)$
 8:     end for
 9:     Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
```

Fig 2. WGAN Algorithm

IV. Result

The expected result would be the improved image generated by both the models with some convergence. However, the model is failing because of some mismatch in the shape of the input in the convolution layer initializations and so the code is not working properly. I am getting some errors because of which the actual result is not available with me.
I am still working on the codes looking for what went wrong with the model.

References

[1] https://gluon.mxnet.io/chapter14_generative-adversarial-networks/dcgan.html

[2] https://gluon.mxnet.io/chapter14_generative-adversarial-networks/dcgan.html

[3] Wasserstein GAN, Martin Arjovsky and Soumith Chintala and Léon Bottou, 2017, https://arxiv.org/abs/1701.07875

[4] How to Develop a Wasserstein Generative Adversarial Network (WGAN) from scratch, by Jason Brownlee, July 17, 2019, https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/

[5] DCGAN, Li Yin, Mar 8, 2017, medium, https://medium.com/@liyin2015/dcgan-79af14a1c247

[6] Image Completion with Deep Learning in Tensorflow, by Brandon Amos, August 9, 2016, http://bamos.github.io/2016/08/09/deep-completion/