CS294-158 Deep Unsupervised Learning
UC Berkeley, Spring 2019
HW4: Generative Adversarial Networks
Due: April 9th, 11:59pm

1 GANs on CIFAR-10

In this exercise, you will train GANs on CIFAR-10. You can download CIFAR 10 from https://www.cs.toronto.edu/~kriz/cifar.html or some other source, such as https://www.tensorflow.org/api_docs/python/tf/keras/datasets/cifar10/load_data.

Start with $z \in \mathbb{R}^{128}$. Use the same architecture prescribed in the SN-GAN [4] CIFAR Resnet. Specifically, the noise is sampled from a standard normal gaussian. The architecture diagrams for the generator and discriminator are described in Figures 1 and 2 respectively. In the generator, for every spatial up-sampling ResBlock, use the nearest neighbor upsampling technique. Specifically, this is implemented as follows:

```
# NHWC: Spatial Upsampling
def upsample_conv2d(x, *, filter_size=(3, 3), n_filters=256):
    _x = tf.concat([x, x, x, x], axis=-1)
    _x = tf.depth_to_space(_x, block_size=2)
    _x = conv2d(_x, filter_size, n_filters, padding='same')
    return _x
```

Similarly, a downsample convolution is implemented by using Spatial Mean Pooling.

```
# NHWC: Spatial Upsampling
def downsample_conv2d(x, *, filter_size=(3, 3), n_filters=256):
    _x = tf.space_to_depth(x, block_size=2)
    _x = tf.add_n(tf.split(_x, 4, axis=-1))/4.
    _x = conv2d(_x, filter_size, n_filters, padding='same')
    return _x
```

The ResBlock Up (or Down) are regular residual blocks except that you will be using an upsample (or downsample) convolution to increase (reduce) the spatial dimension. Note that this is different

from how supervised learning typically reduces spatial dimensions with a strided convolution. As an example, here is how the ResBlock Up (with Batch Norm) is supposed to be implemented (similar implementation for downsampling):

```
# NHWC:
def ResBlockUp(x, *, filter_size=(3, 3), n_filters=256, scope='up_block'):
    with tf.variable_scope(scope):
        _x = BN(x)
        _x = tf.nn.relu(_x)
        _x = conv2d(_x, filter_size, n_filters, padding='same')
        _x = BN(_x)
        _x = tf.nn.relu(_x)
        residual = upsample_conv2d(_x, filter_size, n_filters, padding='same')
        shortcut = upsample_conv2d(x, filter_size=(1, 1), n_filters=n_filters,
                                   padding='same')
        return  shortcut + residual
```

For a regular ResBlock (used in the Discriminator), the shortcut is just the identity connection. To be precise about the generator architecture's implementation, here's a python snippet:

```
# NHWC:
def Generator(*, n_samples=1024):
    with tf.variable_scope('generator'):
        z = tf.random_normal([n_samples, 128])
        out = tf.layers.dense(z, 4 * 4 * 256)
        out = tf.reshape(out, [-1, 4, 4, 256])
        for block in range(3):
            out = ResBlockUp(out, filter_size=(3, 3), n_filters=256,
                             scope=f'res_up_{block}')
        out = BN(out)
        out = tf.nn.relu(out)
        out = conv2d(out, filter_size=(3, 3), n_filters=3)
        out = tf.nn.tanh(out)
        return out
```

Note that as pointed out in the class, Batch Normalization for convolution layers is implemented with a mean over spatial axes as well (in addition to the batch). Also note that the discriminator ResBlocks **do not use batch normalization**. The implementation for the Discriminator must follow the Figure 2.

With the architecture clear and following optimization hyperparameters Adam parameters: $\alpha = 2e-4, \beta_1 = 0, \beta_2 = 0.9$, $\lambda = 10$, $n_{\text{critic}} = 5$, LR (α) decayed to 0 over 100K iterations, implement the following variations of WGAN-GP [2] on CIFAR.

- WGAN-GP, batch size=128. The LR should be decayed to 0 over 100K iterations.

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
dense, $4 \times 4 \times 256$
ResBlock up 256
ResBlock up 256
ResBlock up 256
BN, ReLU, 3×3 conv, 3 Tanh

(a) Generator

Figure 1: Generator

- WGAN-GP, with ResBlocks in generator using 128 filters instead of 256 filters but trained with a larger batch size 256 instead of 128.
- WGAN-GP, with self-modulated batch-normalization in the generator [ie, the gain and bias terms are a function of z at each ResBlockUp]. This is explained in [1].
- (Optional / Bonus): WGAN-GP with instance noise and gradient norm penalities on just generated and real data samples (as opposed to interpolated samples) as in [3].

You are expected to report the following for the above models trained for 50K (or more) iterations:

- 100 samples
- Inception Score (`https://github.com/openai/improved-gan/tree/master/inception_score`) but with an Inception Architecture Classifier trained from scratch on CIFAR-10. Use the same classifier for all the models.
- (Optional / Bonus): Freschet Inception Distance of the final models (with mean and standard deviation of your estimates)

References

[1] Ting Chen et al. "On self modulation for generative adversarial networks". In: *arXiv preprint arXiv:1810.01365* (2018).

[2] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5767–5777.

[3] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. "Which training methods for GANs do actually Converge?" In: *arXiv preprint arXiv:1801.04406* (2018).

[4] Takeru Miyato et al. "Spectral normalization for generative adversarial networks". In: *arXiv preprint arXiv:1802.05957* (2018).

RGB image $x \in \mathbb{R}^{32\times32\times3}$
ResBlock down 128
ResBlock down 128
ResBlock 128
ResBlock 128
ReLU
Global sum pooling
dense $\rightarrow$ 1

(b) Discriminator

Figure 2: Discriminator