dcgan-mnist-v5

June 20, 2024

[2]:
```
#Importing all the necessary libraries
#DGM model implemented using tensorlow keras
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np
import os
```

[3]:
```
#loading the standard MNIST dataset from tensorflow datasets and preprocessing␣
 ↪the image
#the images in the MNIST dataset are 28*28
(train_images, _), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).
 ↪astype('float32')
train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-1, 1]
BUFFER_SIZE = 60000
BATCH_SIZE = 256
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).
 ↪shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

[4]:
```
#generator model using keras sequential
#deconvolution (transposed convolution or upscaling) done to generate image␣
 ↪from latent space vector z of dim 100
# z is upscaled to 28*28 (784) dimensional image using multiple convolution␣
 ↪layers
#batch normalization done to stablise the training of the model
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
```

```
    assert model.output_shape == (None, 7, 7, 256)

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),␣
 ↪padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),␣
 ↪padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',␣
 ↪use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

[5]:
```
#defining the discriminator model
#conv2D to implement convolution layers
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',␣
 ↪input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

[6]:
```
#defining generator and discriminator loss using Binary Cross Entropy
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

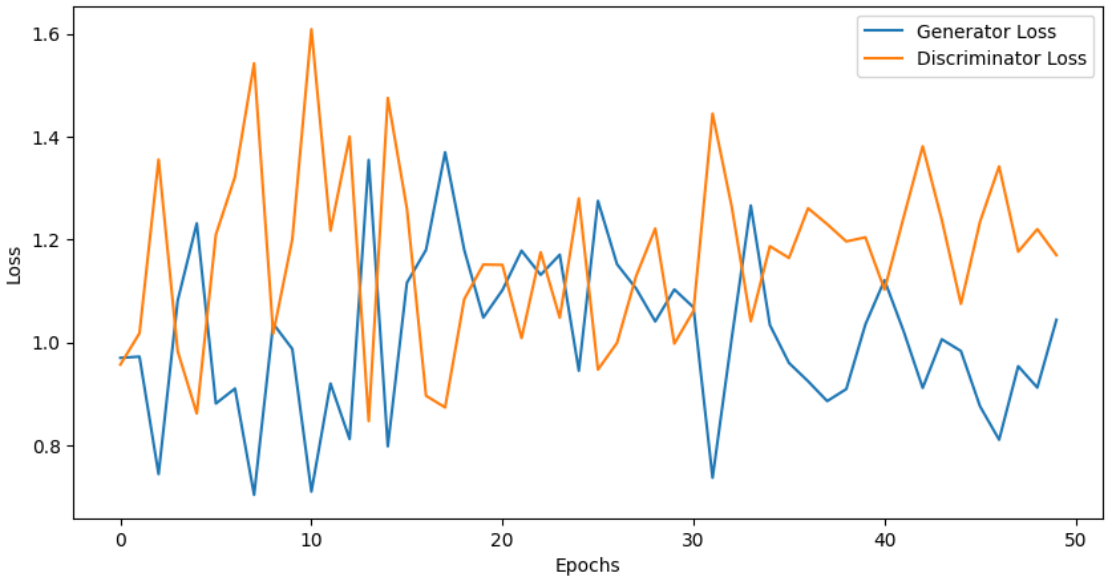
[7]:
```
import time

EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

seed = tf.random.normal([num_examples_to_generate, noise_dim])

@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.
 ↪trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.
 ↪trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.
 ↪trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,␣
 ↪discriminator.trainable_variables))

    return gen_loss, disc_loss

def train(dataset, epochs):
    generator_losses = []
    discriminator_losses = []
    for epoch in range(epochs):
        start = time.time()

        for image_batch in dataset:
            gen_loss, disc_loss = train_step(image_batch)
```

```
        generator_losses.append(gen_loss)
        discriminator_losses.append(disc_loss)

        print (f'Time for epoch {epoch+1} is {time.time()-start} sec')

    return generator_losses, discriminator_losses

generator = make_generator_model()
discriminator = make_discriminator_model()

generator_losses, discriminator_losses = train(train_dataset, EPOCHS)
```

```
Time for epoch 1 is 20.276065349578857 sec
Time for epoch 2 is 9.761974334716797 sec
Time for epoch 3 is 9.911672592163086 sec
Time for epoch 4 is 9.94800066947937 sec
Time for epoch 5 is 10.039191007614136 sec
Time for epoch 6 is 10.138843774795532 sec
Time for epoch 7 is 10.288171529769897 sec
Time for epoch 8 is 10.368274688720703 sec
Time for epoch 9 is 10.51313304901123 sec
Time for epoch 10 is 10.63753366470337 sec
Time for epoch 11 is 10.736661672592163 sec
Time for epoch 12 is 10.700400590896606 sec
Time for epoch 13 is 10.612165451049805 sec
Time for epoch 14 is 10.544219493865967 sec
Time for epoch 15 is 10.513644695281982 sec
Time for epoch 16 is 10.544042348861694 sec
Time for epoch 17 is 10.518793106079102 sec
Time for epoch 18 is 10.546067714691162 sec
Time for epoch 19 is 10.576054334640503 sec
Time for epoch 20 is 10.602833986282349 sec
Time for epoch 21 is 10.608079671859741 sec
Time for epoch 22 is 10.564969539642334 sec
Time for epoch 23 is 10.573159217834473 sec
Time for epoch 24 is 10.561530113220215 sec
Time for epoch 25 is 10.544068336486816 sec
Time for epoch 26 is 10.579376220703125 sec
Time for epoch 27 is 10.552639722824097 sec
Time for epoch 28 is 10.549550533294678 sec
Time for epoch 29 is 10.548936367034912 sec
Time for epoch 30 is 10.540998458862305 sec
Time for epoch 31 is 10.588533163070679 sec
Time for epoch 32 is 10.551876306533813 sec
Time for epoch 33 is 10.554181814193726 sec
Time for epoch 34 is 10.552859544754028 sec
```

```
Time for epoch 35 is 10.581958770751953 sec
Time for epoch 36 is 10.601291179656982 sec
Time for epoch 37 is 10.557964324951172 sec
Time for epoch 38 is 10.56647276878357 sec
Time for epoch 39 is 10.561517477035522 sec
Time for epoch 40 is 10.58103322982788 sec
Time for epoch 41 is 10.57373046875 sec
Time for epoch 42 is 10.554275512695312 sec
Time for epoch 43 is 10.563316345214844 sec
Time for epoch 44 is 10.607719898223877 sec
Time for epoch 45 is 10.588618040084839 sec
Time for epoch 46 is 10.55614709854126 sec
Time for epoch 47 is 10.557905435562134 sec
Time for epoch 48 is 10.562829971313477 sec
Time for epoch 49 is 10.590407848358154 sec
Time for epoch 50 is 10.602385520935059 sec
```

[8]:
```python
plt.figure(figsize=(10, 5))
plt.plot(generator_losses, label='Generator Loss')
plt.plot(discriminator_losses, label='Discriminator Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

min_error = min(generator_losses + discriminator_losses)
print(f"Minimum Error Observed: {min_error.numpy()}")
```



```
Minimum Error Observed: 0.7031219005584717
```

[9]:
```
generator.save('generator_model.h5')
discriminator.save('discriminator_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
```

[11]:
```
generator.save('generator_model.keras')
discriminator.save('discriminator_model.keras')
```

[ ]:
```

```

[ ]:
```

```

[12]:
```
generator = tf.keras.models.load_model('generator_model.keras')
discriminator = tf.keras.models.load_model('discriminator_model.keras')
```

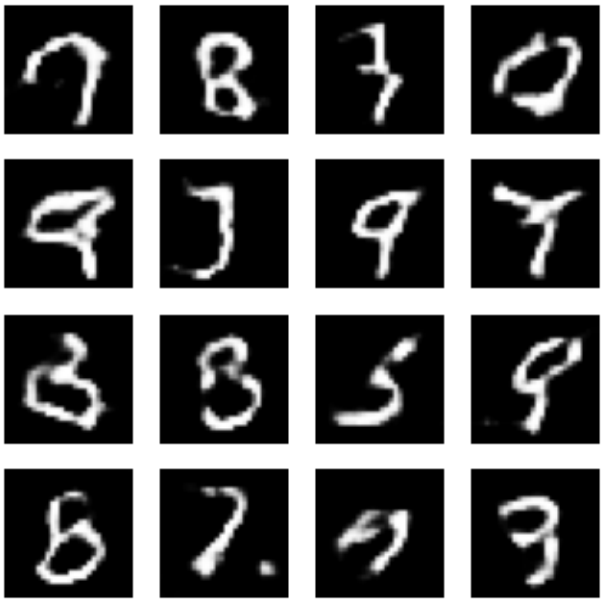
[14]:
```
# Plot the generated images
def plot_images(images):
    fig = plt.figure(figsize=(4, 4))
    for i in range(images.shape[0]):
        plt.subplot(4, 4, i + 1)
        plt.imshow((images[i, :, :, 0] * 127.5 + 127.5).numpy().
 ↪astype("uint8"), cmap='gray')
        plt.axis('off')
    plt.show()
```

[15]:
```
num_examples_to_generate = 16
noise_dim = 100
random_noise = tf.random.normal([num_examples_to_generate, noise_dim])
generated_images = generator(random_noise, training=False)

# Pass the generated images through the discriminator
discriminator_output = discriminator(generated_images, training=False)

# Calculate the error
```

```
errors = tf.keras.losses.binary_crossentropy(tf.
  ↪ones_like(discriminator_output), discriminator_output)
average_error = tf.reduce_mean(errors)
print(f"Average Discriminator Error on Generated Images: {average_error.
  ↪numpy()}")
```

```
Average Discriminator Error on Generated Images: 10.243393898010254
```

[16]:
```
plot_images(generated_images)
```



[17]:
```
# Plot the error
plt.figure(figsize=(6, 4))
plt.bar(range(num_examples_to_generate), errors.numpy())
plt.xlabel('Image Index')
plt.ylabel('Error')
plt.title('Discriminator Error for Generated Images')
plt.show()
```


Discriminator Error for Generated Images
Error
Image Index
0
2
4
6
8
10
12
14
16