dcgan-minst-v5-more-layers

June 20, 2024

[2]:
```
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np
import os
```

[3]:
```
(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
```

[18]:
```
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(256, (5, 5), strides=(1, 1),␣
 ↪padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(2, 2),␣
 ↪padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 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, 28, 28, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(32, (5, 5), strides=(1, 1),␣
↪padding='same', use_bias=False))
    assert model.output_shape == (None, 28, 28, 32)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

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

    return model
```

[19]:
```
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.Conv2D(256, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(512, (5, 5), strides=(1, 1), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

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

    return model
```

[20]:
```
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)
```

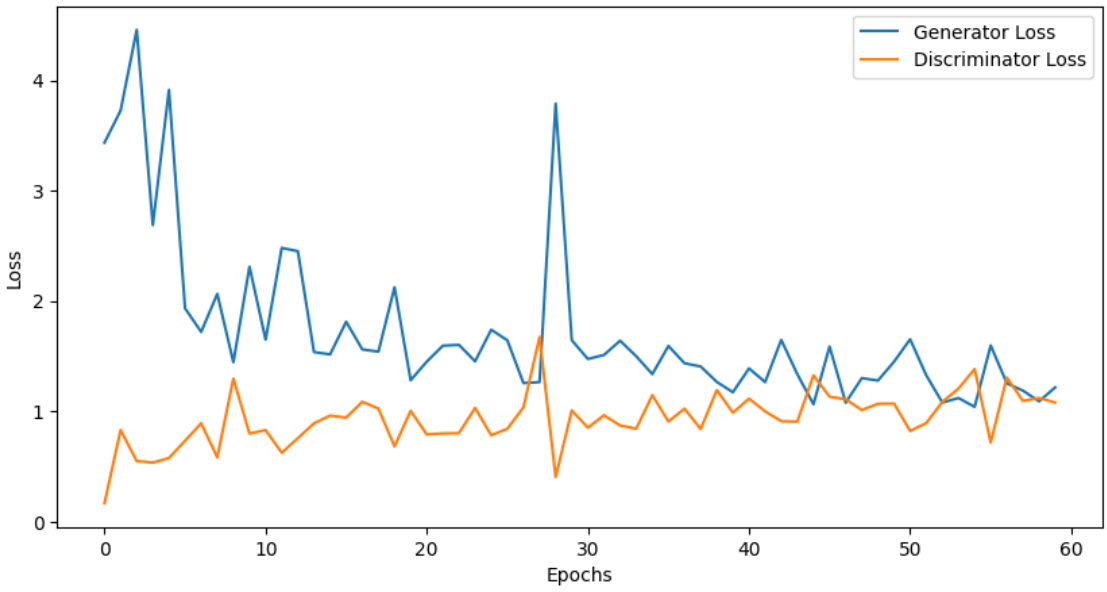
[21]:
```
import time

EPOCHS = 60
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 57.040189266204834 sec
Time for epoch 2 is 43.445393562316895 sec
Time for epoch 3 is 43.74435591697693 sec
Time for epoch 4 is 43.645939350128174 sec
Time for epoch 5 is 43.59630632400513 sec
Time for epoch 6 is 43.59605360031128 sec
Time for epoch 7 is 43.62102437019348 sec
Time for epoch 8 is 43.656681060791016 sec
Time for epoch 9 is 43.680811405181885 sec
Time for epoch 10 is 43.68891906738281 sec
Time for epoch 11 is 43.703463315963745 sec
Time for epoch 12 is 43.69803333282471 sec
Time for epoch 13 is 43.703205585479736 sec
Time for epoch 14 is 43.69241189956665 sec
Time for epoch 15 is 43.69132995605469 sec
Time for epoch 16 is 43.667670249938965 sec
Time for epoch 17 is 43.6682915687561 sec
Time for epoch 18 is 43.68287968635559 sec
Time for epoch 19 is 43.68759846687317 sec
Time for epoch 20 is 43.68545985221863 sec
Time for epoch 21 is 43.684765338897705 sec
Time for epoch 22 is 43.817991971969604 sec
Time for epoch 23 is 43.62626624107361 sec
Time for epoch 24 is 43.66624474525452 sec
Time for epoch 25 is 43.81125831604004 sec
Time for epoch 26 is 43.60022258758545 sec
Time for epoch 27 is 43.665045499801636 sec
Time for epoch 28 is 43.673073053359985 sec
Time for epoch 29 is 43.80473732948303 sec
```

```
Time for epoch 30 is 43.59255385398865 sec
Time for epoch 31 is 43.672481298446655 sec
Time for epoch 32 is 43.674423694610596 sec
Time for epoch 33 is 43.675015926361084 sec
Time for epoch 34 is 43.65449500083923 sec
Time for epoch 35 is 43.64660334587097 sec
Time for epoch 36 is 43.609994649887085 sec
Time for epoch 37 is 43.59799861907959 sec
Time for epoch 38 is 43.61752724647522 sec
Time for epoch 39 is 43.642062187194824 sec
Time for epoch 40 is 43.68002891540527 sec
Time for epoch 41 is 43.71491503715515 sec
Time for epoch 42 is 43.70704197883606 sec
Time for epoch 43 is 43.81472635269165 sec
Time for epoch 44 is 43.60007858276367 sec
Time for epoch 45 is 43.69814419746399 sec
Time for epoch 46 is 43.68024969100952 sec
Time for epoch 47 is 43.67201042175293 sec
Time for epoch 48 is 43.64000725746155 sec
Time for epoch 49 is 43.631139516830444 sec
Time for epoch 50 is 43.622344732284546 sec
Time for epoch 51 is 43.600080490112305 sec
Time for epoch 52 is 43.60224747657776 sec
Time for epoch 53 is 43.60750699043274 sec
Time for epoch 54 is 43.62879467010498 sec
Time for epoch 55 is 43.67660093307495 sec
Time for epoch 56 is 43.72691345214844 sec
Time for epoch 57 is 43.899224519729614 sec
Time for epoch 58 is 43.61165142059326 sec
Time for epoch 59 is 43.66374707221985 sec
Time for epoch 60 is 43.66728067398071 sec
```

[22]:
```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: 0.16659587621688843
```

[23]:
```
generator.save('generator_model.h5')
discriminator.save('discriminator_model.h5')
```

```
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.
```

[24]:
```
generator.save('generator_model.keras')
discriminator.save('discriminator_model.keras')
```

[ ]:

[ ]:

[25]:
```
generator = tf.keras.models.load_model('generator_model.keras')
discriminator = tf.keras.models.load_model('discriminator_model.keras')
```

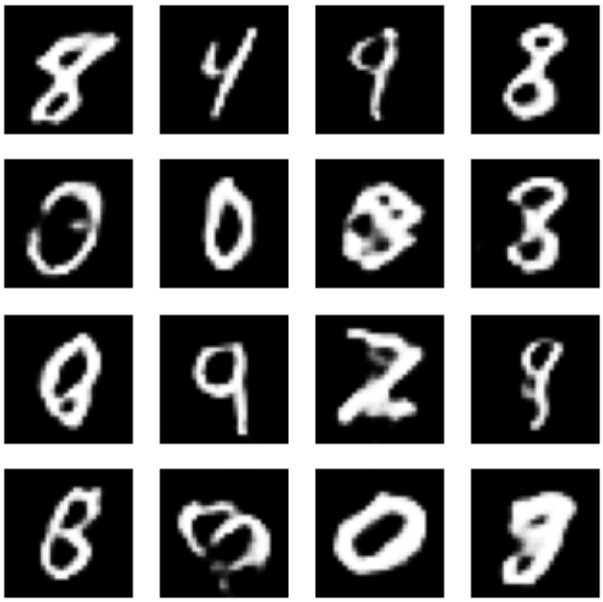
[26]:
```
# Plot the generated images
def plot_images(images):
    fig = plt.figure(figsize=(4, 4))
    for i in range(images.shape[0]):
        plt.subplot(4, 4, i + 1)
```

```
        plt.imshow((images[i, :, :, 0] * 127.5 + 127.5).numpy().
 ↪astype("uint8"), cmap='gray')
        plt.axis('off')
    plt.show()
```

[27]:
```
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: 11.069901466369629
```

[28]:
```
plot_images(generated_images)
```



[29]:
```
# 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()
```