Charlie Misbach

CSCI-3470

Assignment #4

Due 4/21/2025

1. Runnable Code:

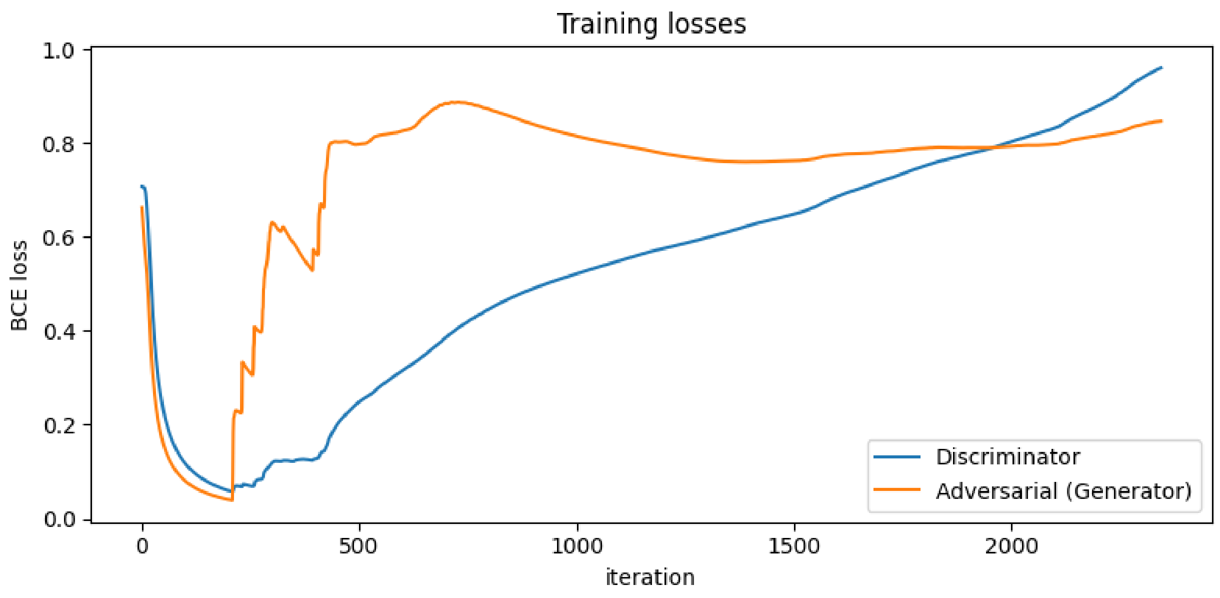
```
# Question: Implement a GAN to generate MNIST
# images, discuss your discovery/thinking during
# the implementation

# 1. Discriminator
from tensorflow import keras
from tensorflow.keras import layers

def make_discriminator():
    return keras.Sequential([
        layers.Input(shape=(28, 28, 1)),                       # 28×28 gray MNIST
        layers.Conv2D(32, 5, strides=2, padding='same'),
        layers.LeakyReLU(0.2),
        layers.Dropout(0.4),

        layers.Conv2D(64, 5, strides=2, padding='same'),
        layers.LeakyReLU(0.2),
        layers.Dropout(0.4),

        layers.Conv2D(128, 5, strides=2, padding='same'),
        layers.LeakyReLU(0.2),
        layers.Dropout(0.4),

        layers.Conv2D(256, 5, strides=2, padding='same'),
        layers.LeakyReLU(0.2),
        layers.Dropout(0.4),

        layers.Flatten(),
        layers.Dense(1, activation='sigmoid') # “real” vs “fake”
    ], name="discriminator")

# 2. Generator

LATENT_DIM = 100          # z-vector length

def make_generator():
    return keras.Sequential([
        layers.Input(shape=(LATENT_DIM,)),
        layers.Dense(7 * 7 * 192, use_bias=False),
        layers.BatchNormalization(),
        layers.ReLU(),
        layers.Reshape((7, 7, 192)),
        layers.Dropout(0.4),

        layers.UpSampling2D(),
        layers.Conv2DTranspose(96, 5, strides=1, padding="same", use_bias=False),
        layers.BatchNormalization(),
        layers.ReLU(),

        layers.UpSampling2D(),
        layers.Conv2DTranspose(48, 5, strides=1, padding="same", use_bias=False),
        layers.BatchNormalization(),
        layers.ReLU(),

        layers.Conv2DTranspose(24, 5, strides=1, padding="same", use_bias=False),
        layers.BatchNormalization(),
        layers.ReLU(),

        layers.Conv2DTranspose(1, 5, strides=1, padding="same", activation="sigmoid")
    ], name="generator")

g = make_generator()
g.summary(line_length=80)

# 3. Implement the algorithm discussed in class to train
# this GAN (Lecture 14). Explore a good training iteration/
# epoch number
import tensorflow as tf, numpy as np, matplotlib.pyplot as plt, pathlib, time

1. Building of the two networks
d = make_discriminator()
g = make_generator()

# compile discriminator alone
d.compile(optimizer=tf.keras.optimizers.Adam(2e-4, 0.5),
          loss="binary_crossentropy",
          metrics=["accuracy"])

# freeze D inside the combined “adversarial” model
d.trainable = False
z_in   = tf.keras.Input(shape=(100,))
img    = g(z_in)
valid  = d(img)
adv    = tf.keras.Model(z_in, valid, name="adv")
adv.compile(optimizer=tf.keras.optimizers.Adam(2e-4, 0.5),
            loss="binary_crossentropy")
d.trainable = True          # make it trainable again for the solo step
```

```
##############################################################################
# 2. data
(train_x, _), _ = tf.keras.datasets.mnist.load_data()
train_x = train_x.astype("float32") / 255. # 0-1
train_x = np.expand_dims(train_x, -1)

BATCH  = 128
data   = tf.data.Dataset.from_tensor_slices(train_x).shuffle(60000).batch(BATCH)
##############################################################################
# 3. training loop
EPOCHS         = 5                                 # number of epochs
SAVE_ITERS     = {1, 200, 400, 600}               # iterations to snapshot
fixed_noise    = tf.random.normal([16, 100])       # same seed every time
outdir         = pathlib.Path("gan_simple"); outdir.mkdir(exist_ok=True)

d_losses, adv_losses = [], []
iteration = 0

for epoch in range(EPOCHS):
    for real in data:
        bs = real.shape[0]
        noise = tf.random.normal([bs, 100])
        fake  = g(noise, training=False)

        # train discriminator
        d_loss_real = d.train_on_batch(real,  np.ones((bs,1)))
        d_loss_fake = d.train_on_batch(fake,  np.zeros((bs,1)))
        d_loss      = 0.5*(d_loss_real[0] + d_loss_fake[0])

        # train generator (via adv model)
        noise = tf.random.normal([bs, 100])
        adv_loss = adv.train_on_batch(noise, np.ones((bs,1)))

        # book-keeping
        d_losses.append(d_loss)
        adv_losses.append(adv_loss)
        iteration += 1

        # snapshot images if this iteration is in SAVE_ITERS
        if iteration in SAVE_ITERS:
            imgs = g(fixed_noise, training=False).numpy()
            imgs = imgs.reshape(4,4,28,28).transpose(0,2,1,3).reshape(4*28,4*28)

            plt.imsave(outdir/f"samples_{iteration:04d}.png",
                       imgs, cmap="gray")

    print(f"Epoch {epoch+1:02}/{EPOCHS} | D={d_loss:.3f}  ADV={adv_loss:.3f}")
```

```
##############################################################################
# 4. loss curves
plt.figure(figsize=(8,4))
plt.plot(d_losses,  label="Discriminator")
plt.plot(adv_losses,label="Adversarial (Generator)")
plt.title("Training losses")
plt.xlabel("iteration")
plt.ylabel("BCE loss")
plt.legend()
plt.tight_layout()
plt.savefig(outdir/"loss_curve.png")
plt.show()

##############################################################################
# 5. show the 4 saved grids
from PIL import Image
for it in sorted(SAVE_ITERS):
    path = outdir / f"samples_{it:04d}.png"
    if path.exists():
        Image.open(path).resize((224, 224)).show()
```

2. Two types of expected results:

(i) display the loss values (loss of the Discriminator, and loss of the entire adversarial model) at each iteration/epoch as a plotted curve;

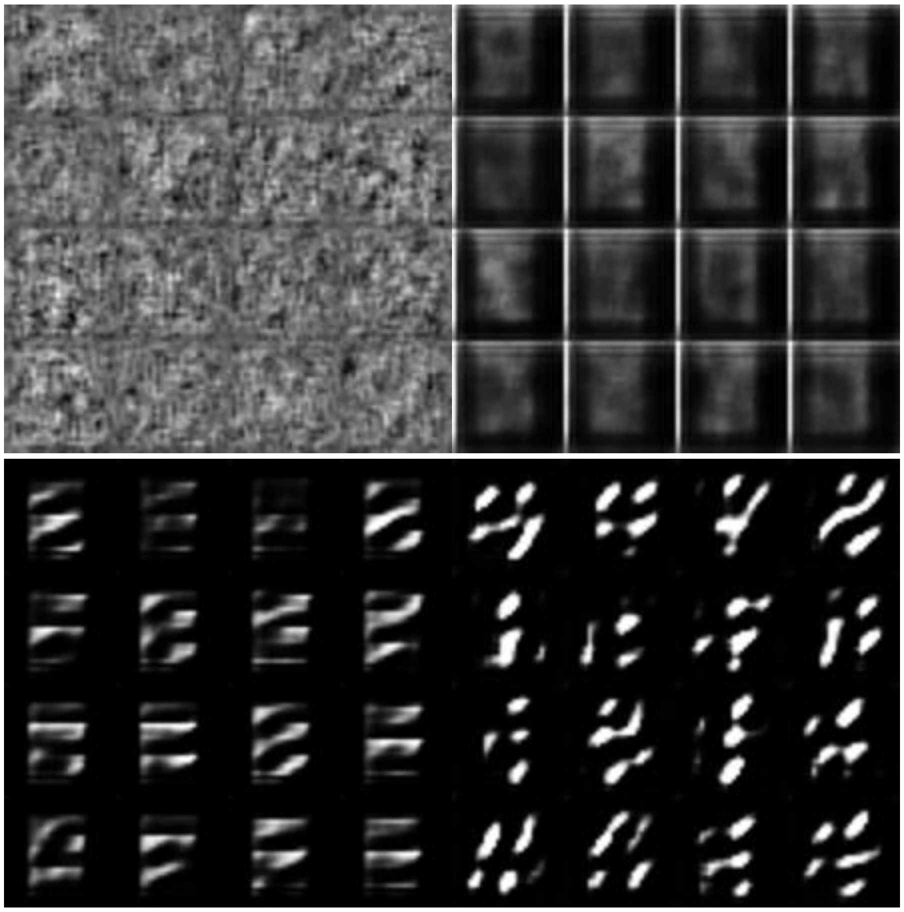


This is a graph of the training loss over 2000 iterations of the dataset for the generator and the discriminator. The X-axis is the number of training steps (1 iteration being 1 batch of images), the Y-axis is the loss value, the lower the better for the generator and discriminator.

As seen in the loss graph, both the generator and discriminator started with high loss values, which is expected at the beginning of training. They quickly improved during the early iterations, and performance appeared to peak at around 300 iterations. After that point, the discriminator seemed to begin overfitting, making training less stable. As a result, both losses started increasing again, indicating that the generator was struggling to keep up. Although I ran training past 2000 iterations out of curiosity, the most effective training likely occurred much earlier

(ii) display at least four groups of generated images (suggest 16 images per group), where each group should be plotted at a certain iteration/epoch to reflect the training (e.g., you may plot a group at iteration 1, 200, 400, and 600).

(1, 200, 400, 600) (top left, top right, bottom left, bottom right) respectively



3. Discussion and thinking discovery:

During the implementation, I saw how sensitive GAN training can be. At first, the generator and discriminator improved quickly, but after a point, the balance between them became unstable. I learned that if the discriminator gets too strong, the generator struggles to improve, which can lead to rising loss values for both. This helped me understand why GANs are hard to train and how important it is to monitor loss trends and stop training at the right time.