CS156 (Introduction to AI), Spring 2021

Homework Assignment #11 submission

Student Name: HUY NGUYEN

Student ID: 015207465

Email address: huy.l.nguyen@sjsu.edu

Solution

```
import numpy as np
from tensorflow import keras
from tensorflow.keras.datasets.mnist import load_data
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Conv2DTranspose
from numpy import expand_dims
from numpy import ones
from numpy import zeros
from numpy.random import rand
from numpy.random import randint
from numpy.random import randn
from numpy import vstack
from numpy import asarray
```

```

(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
input_shape = (28, 28, 1)

#combine into a single dataset
mnist = np.concatenate([x_train, x_test], axis=0)
mnist = expand_dims(mnist, axis=-1)

# Scale images to the [0, 1] range
mnist = mnist.astype("float32") / 255
```

```

mnist.shape
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
32768/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
26427392/26421880 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-l
8192/5148 [==============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-i
4423680/4422102 [==============================] - 0s 0us/step
(70000, 28, 28, 1)
```

```
for i in range(25):
    plt.subplot(5, 5, 1 + i)
    plt.axis('off')
    plt.imshow(x_train[i], cmap='gray')
plt.show()
```

```
# plot reverse gray scale:
for i in range(25):
    plt.subplot(5, 5, 1 + i)
    plt.axis('off')
    plt.imshow(x_train[i], cmap='gray_r')
plt.show()
```

```
# define the standalone discriminator model
def define_discriminator(in_shape=(28, 28, 1)):
    model = Sequential()
    model.add(Conv2D(64, (3, 3), strides=(2, 2), padding='same', input_shape=in_shape))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Conv2D(64, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model


# define the discriminator model
discriminator = define_discriminator()
discriminator.summary()

```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 64)        640
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 3137
=================================================================
Total params: 40,705
Trainable params: 40,705
Non-trainable params: 0
_________________________________________________________________
```

```
# define the standalone generator model
def define_generator(latent_dim):
    model = Sequential()
    # foundation for 7x7 image
```

```
    n_nodes = 128 * 7 * 7
    model.add(Dense(n_nodes, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((7, 7, 128)))
    # upsample to 14x14
    model.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 28x28
    model.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2D(1, (7, 7), activation='sigmoid', padding='same'))
    return model


# size of the latent space
latent_dim = 100
# define the discriminator model
generator = define_generator(latent_dim)
generator.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6272)              633472
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 6272)              0
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 128)       262272
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 128)       262272
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 1)         6273
=================================================================
Total params: 1,164,289
Trainable params: 1,164,289
Non-trainable params: 0
_________________________________________________________________
```

```
# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
```

```
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model


gan_model = define_gan(generator, discriminator)
gan_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (None, 28, 28, 1)         1164289
_________________________________________________________________
sequential (Sequential)      (None, 1)                 40705
=================================================================
Total params: 1,204,994
Trainable params: 1,164,289
Non-trainable params: 40,705
_________________________________________________________________
```

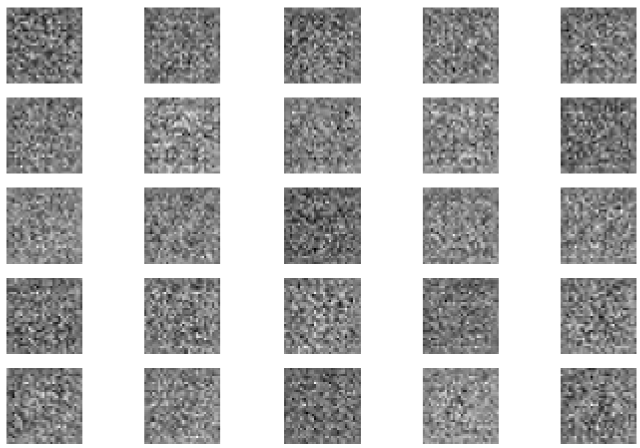
```
# without training, our generator model produces really bad images (they are not very good

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input


# use the generator to generate n fake examples, with class labels
def generate_fake_generator_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # create 'fake' class labels (0)
    y = zeros((n_samples, 1))
    return X, y


# generate samples
n_samples = 25
X, _ = generate_fake_generator_samples(generator, latent_dim, n_samples)
# plot the generated samples
for i in range(n_samples):
    # define subplot
    plt.subplot(5, 5, 1 + i)
```

```
    plt.subplot(5, 5, 1 + i)
    # turn off axis labels
    plt.axis('off')
    # plot single image
    plt.imshow(X[i, :, :, 0], cmap='gray_r')
# show the figure
plt.show()
```



```
# select real samples
def generate_real_samples(dataset, n_samples):
    # choose random instances
    ix = randint(0, dataset.shape[0], n_samples)
    # retrieve selected images
    X = dataset[ix]
    # generate 'real' class labels (1)
    y = ones((n_samples, 1))
    return X, y


# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # create 'fake' class labels (0)
    y = zeros((n_samples, 1))
    return X, y


# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

```

```
# evaluate the discriminator, plot generated images, save generator model
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples=100):
    # prepare real samples
    X_real, y_real = generate_real_samples(dataset, n_samples)
    # evaluate discriminator on real examples
    _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
    # evaluate discriminator on fake examples
    _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real * 100, acc_fake * 100))
    # save plot
    #save_plot(x_fake, epoch)
    # save the generator model tile file
    #filename = 'generator_model_%03d.h5' % (epoch + 1)
    #g_model.save(filename)  # serializing the model: https://www.tensorflow.org/tutorials


# train the generator and discriminator together
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100, n_batch=256):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    # manually enumerate epochs
    for i in range(n_epochs):
        # enumerate batches over the training set
        for j in range(bat_per_epo):
            # get randomly selected 'real' samples
            X_real, y_real = generate_real_samples(dataset, half_batch)
            # generate 'fake' examples
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
            # create training set for the discriminator
            X, y = vstack((X_real, X_fake)), vstack((y_real, y_fake))
            # update discriminator model weights
            d_loss, _ = d_model.train_on_batch(X, y)
            # prepare points in latent space as input for the generator
            X_gan = generate_latent_points(latent_dim, n_batch)
            # create inverted labels for the fake samples
            y_gan = ones((n_batch, 1))
            # update the generator via the discriminator's error
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
            # summarize loss on this batch
            print('>%d, %d/%d, d_loss=%.3f, g_loss=%.3f' % (i + 1, j + 1, bat_per_epo, d_l
        # evaluate the model performance, sometimes
        #if (i+1) % 10 == 0:
    summarize_performance(i, g_model, d_model, dataset, latent_dim)

    return g_model
```

```
# size of the latent space
```

```
latent_dim = 100
# train model
trained_generator = train(generator, discriminator, gan_model, mnist, latent_dim, 30)
```

```
>30, 161/273, d_loss=0.690, g_loss=0.694
>30, 162/273, d_loss=0.685, g_loss=0.709
>30, 163/273, d_loss=0.691, g_loss=0.726
>30, 164/273, d_loss=0.693, g_loss=0.742
>30, 165/273, d_loss=0.689, g_loss=0.751
>30, 166/273, d_loss=0.688, g_loss=0.748
>30, 167/273, d_loss=0.692, g_loss=0.729
>30, 168/273, d_loss=0.695, g_loss=0.705
>30, 169/273, d_loss=0.691, g_loss=0.679
>30, 170/273, d_loss=0.688, g_loss=0.660
>30, 171/273, d_loss=0.691, g_loss=0.653
>30, 172/273, d_loss=0.686, g_loss=0.658
>30, 173/273, d_loss=0.696, g_loss=0.683
>30, 174/273, d_loss=0.692, g_loss=0.703
>30, 175/273, d_loss=0.696, g_loss=0.722
>30, 176/273, d_loss=0.693, g_loss=0.740
>30, 177/273, d_loss=0.693, g_loss=0.743
>30, 178/273, d_loss=0.696, g_loss=0.744
>30, 179/273, d_loss=0.688, g_loss=0.739
>30, 180/273, d_loss=0.695, g_loss=0.718
>30, 181/273, d_loss=0.699, g_loss=0.703
>30, 182/273, d_loss=0.690, g_loss=0.692
>30, 183/273, d_loss=0.692, g_loss=0.670
>30, 184/273, d_loss=0.684, g_loss=0.663
>30, 185/273, d_loss=0.686, g_loss=0.658
>30, 186/273, d_loss=0.692, g_loss=0.666
>30, 187/273, d_loss=0.694, g_loss=0.683
>30, 188/273, d_loss=0.693, g_loss=0.720
>30, 189/273, d_loss=0.698, g_loss=0.749
>30, 190/273, d_loss=0.685, g_loss=0.762
>30, 191/273, d_loss=0.695, g_loss=0.746
>30, 192/273, d_loss=0.689, g_loss=0.735
>30, 193/273, d_loss=0.691, g_loss=0.719
>30, 194/273, d_loss=0.686, g_loss=0.687
>30, 195/273, d_loss=0.689, g_loss=0.667
>30, 196/273, d_loss=0.693, g_loss=0.649
>30, 197/273, d_loss=0.686, g_loss=0.661
>30, 198/273, d_loss=0.694, g_loss=0.675
>30, 199/273, d_loss=0.687, g_loss=0.693
>30, 200/273, d_loss=0.691, g_loss=0.698
>30, 201/273, d_loss=0.690, g_loss=0.726
>30, 202/273, d_loss=0.688, g_loss=0.732
>30, 203/273, d_loss=0.692, g_loss=0.739
>30, 204/273, d_loss=0.690, g_loss=0.730
>30, 205/273, d_loss=0.692, g_loss=0.720
>30, 206/273, d_loss=0.696, g_loss=0.695
>30, 207/273, d_loss=0.692, g_loss=0.684
>30, 208/273, d_loss=0.693, g_loss=0.681
>30, 209/273, d_loss=0.685, g_loss=0.683
>30, 210/273, d_loss=0.687, g_loss=0.693
>30, 211/273, d_loss=0.692, g_loss=0.689
>30, 212/273, d_loss=0.698, g_loss=0.690
>30, 213/273, d_loss=0.691, g_loss=0.688
```

```
>30, 214/273, d_loss=0.697, g_loss=0.693
>30, 215/273, d_loss=0.690, g_loss=0.696
>30, 216/273, d_loss=0.696, g_loss=0.710
>30, 217/273, d_loss=0.694, g_loss=0.715
>30, 218/273, d_loss=0.690, g_loss=0.740
>30, 219/273, d_loss=0.692, g_loss=0.730
```

```
#### 30 epochs

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input


# create and display a plot of generated images (reversed grayscale)
def display_plot(examples, n):
    for i in range(n * n):
        plt.subplot(n, n, 1 + i)
        plt.axis('off')
        plt.imshow(examples[i, :, :, 0], cmap='gray_r')
    plt.show()


# load model
#model = load_model('generator_model_100.h5') #load the last seralized model (latest versi
# generate images
latent_points = generate_latent_points(100, 25)
# generate images
X = trained_generator.predict(latent_points)
# plot the result
display_plot(X, 5)
```

```

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
```

```
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input


# create and display a plot of generated images (reversed grayscale)
def display_plot(examples, n):
    for i in range(n * n):
        plt.subplot(n, n, 1 + i)
        plt.axis('off')
        plt.imshow(examples[i, :, :, 0], cmap='gray_r')
    plt.show()


# load model
#model = load_model('generator_model_100.h5') #load the last seralized model (latest versi
# generate images
latent_points = generate_latent_points(100, 25)
# generate images
X = trained_generator.predict(latent_points)
# plot the result
display_plot(X, 5)
```

0s completed at 6:24 PM