WuBenjaminAssignment10

May 8, 2022

1 CS156 (Introduction to AI), Spring 2022

2 Homework 10 submission

2.0.1 Roster Name: Benjamin Wu

2.0.2 Student ID: 013607880

2.0.3 Email address: benjamin.wu01@sjsu.edu

2.1 Solution

2.2 Import libraries, setup random seed

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

[ ]:
```
np.random.seed(42)
```

2.3 References and sources

List all your references and sources here. This includes all sites/discussion boards/blogs/posts/etc. where you grabbed some code examples.

2.4 Code the solution

[ ]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

x = np.concatenate((x_train, x_test))
y = np.concatenate((y_train, y_test))

x = expand_dims(x, axis=-1)

x = x.astype("float32") / 255

x.shape
```

[ ]:
```
(70000, 28, 28, 1)
```

[ ]:
```
# 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(Conv2D(64, (5,5), strides=(1, 1), 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
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 64)          102464
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 7, 7, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 3137
=================================================================
Total params: 143,169
Trainable params: 143,169
Non-trainable params: 0
_________________________________________________________________
C:\Users\benja\anaconda3\envs\test\lib\site-
packages\keras\optimizer_v2\optimizer_v2.py:355: UserWarning: The `lr` argument
is deprecated, use `learning_rate` instead.
  warnings.warn(
```

[ ]:
```python
# 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))
    model.add(Conv2DTranspose(128, (1,1), strides=(1,1), 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 generator 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_3 (LeakyReLU)    (None, 6272)              0
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 128)       262272
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 128)       16512
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 128)       262272
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 1)         6273
=================================================================
Total params: 1,180,801
Trainable params: 1,180,801
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
    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)         1180801
_________________________________________________________________
sequential (Sequential)      (None, 1)                 143169
=================================================================
Total params: 1,323,970
Trainable params: 1,180,801
Non-trainable params: 143,169
_________________________________________________________________
```

[ ]:
```
# 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/keras/save_and_load

# 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_loss, g_loss))
        # 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, x, latent_dim,␣
 ↪10)
```

[ ]:
```
# 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 version of the GAN model)
# generate images
latent_points = generate_latent_points(100, 25)
# generate images
X = trained_generator.predict(latent_points)
# plot the result
display_plot(X, 5)
```