4 Additional Functions with Keras 2

- 4.1 LSTM with MNIST
- 4.2 GAN with MNIST

4.1 LSTM with MNIST

(1) 데이터셋

In [1]:
```
import tensorflow.keras.utils as utils
from tensorflow.keras import datasets
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense

import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
(X_train, Y_train),(X_test, Y_test) = datasets.mnist.load_data()
X_train_norm = X_train.astype('float32')/255.0
X_test_norm = X_test.astype('float32')/255.0
Y_train_onehot = utils.to_categorical(Y_train)
Y_test_onehot = utils.to_categorical(Y_test)

print(X_train_norm.shape, Y_train_onehot.shape)
n_in = X_train.shape[1:]
n_out = Y_train_onehot.shape[-1]
```

```
(60000, 28, 28) (60000, 10)
```

(2) 모델링

In [3]:
```
def lstm(n_in, n_out):
    # Coding Time
    model = Sequential()
    model.add(LSTM(30, input_shape=n_in))
    model.add(Dense(n_out, activation='softmax'))
    return model

model = lstm(n_in, n_out)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30)                7080
_________________________________________________________________
dense (Dense)                (None, 10)                310
=================================================================
Total params: 7,390
Trainable params: 7,390
Non-trainable params: 0
_________________________________________________________________
```

(3) 모델의 학습과정 설정

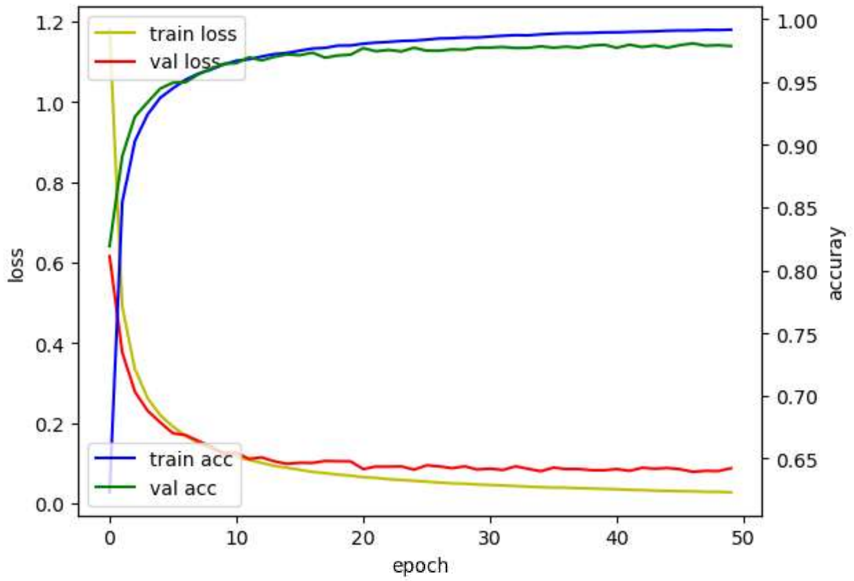
In [4]:
```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

(4) 모델 학습

In [5]:
```
from tensorflow.keras.callbacks import EarlyStopping
# Coding Time
earlystopper = EarlyStopping(monitor='val_accuracy', patience=7, verbose=1, mode='auto', restore_best_weights=True)
history = model.fit(X_train_norm, Y_train_onehot, batch_size=128, epochs=50, validation_split=0.2, callbacks = [earlystopper])
```

```
Epoch 1/50
375/375 [==============================] - 5s 8ms/step - loss: 1.1792 - accuracy: 0.6231 - val_loss: 0.6155 - val_accuracy: 0.8192
Epoch 2/50
375/375 [==============================] - 2s 6ms/step - loss: 0.4904 - accuracy: 0.8541 - val_loss: 0.3766 - val_accuracy: 0.8913
Epoch 3/50
375/375 [==============================] - 2s 6ms/step - loss: 0.3345 - accuracy: 0.9032 - val_loss: 0.2773 - val_accuracy: 0.9226
Epoch 4/50
375/375 [==============================] - 2s 6ms/step - loss: 0.2614 - accuracy: 0.9243 - val_loss: 0.2302 - val_accuracy: 0.9339
Epoch 5/50
375/375 [==============================] - 2s 6ms/step - loss: 0.2190 - accuracy: 0.9375 - val_loss: 0.2008 - val_accuracy: 0.9448
Epoch 6/50
375/375 [==============================] - 2s 7ms/step - loss: 0.1907 - accuracy: 0.9450 - val_loss: 0.1739 - val_accuracy: 0.9495
Epoch 7/50
375/375 [==============================] - 2s 6ms/step - loss: 0.1671 - accuracy: 0.9518 - val_loss: 0.1695 - val_accuracy: 0.9499
Epoch 8/50
375/375 [==============================] - 2s 6ms/step - loss: 0.1500 - accuracy: 0.9567 - val_loss: 0.1549 - val_accuracy: 0.9561
Epoch 9/50
375/375 [==============================] - 2s 6ms/step - loss: 0.1361 - accuracy: 0.9599 - val_loss: 0.1406 - val_accuracy: 0.9607
Epoch 10/50
375/375 [==============================] - 2s 6ms/step - loss: 0.1239 - accuracy: 0.9640 - val_loss: 0.1242 - val_accuracy: 0.9645
Epoch 11/50
375/375 [==============================] - 2s 6ms/step - loss: 0.1146 - accuracy: 0.9669 - val_loss: 0.1250 - val_accuracy: 0.9650
Epoch 12/50
375/375 [==============================] - 2s 6ms/step - loss: 0.1077 - accuracy: 0.9681 - val_loss: 0.1103 - val_accuracy: 0.9696
Epoch 13/50
375/375 [==============================] - 2s 6ms/step - loss: 0.1002 - accuracy: 0.9703 - val_loss: 0.1140 - val_accuracy: 0.9672
Epoch 14/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0928 - accuracy: 0.9722 - val_loss: 0.1041 - val_accuracy: 0.9700
Epoch 15/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0881 - accuracy: 0.9731 - val_loss: 0.0980 - val_accuracy: 0.9719
Epoch 16/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0830 - accuracy: 0.9749 - val_loss: 0.1006 - val_accuracy: 0.9712
Epoch 17/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0780 - accuracy: 0.9766 - val_loss: 0.1003 - val_accuracy: 0.9733
Epoch 18/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0748 - accuracy: 0.9771 - val_loss: 0.1051 - val_accuracy: 0.9693
Epoch 19/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0711 - accuracy: 0.9789 - val_loss: 0.1044 - val_accuracy: 0.9711
Epoch 20/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0686 - accuracy: 0.9790 - val_loss: 0.1042 - val_accuracy: 0.9717
Epoch 21/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0647 - accuracy: 0.9804 - val_loss: 0.0849 - val_accuracy: 0.9769
Epoch 22/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0629 - accuracy: 0.9813 - val_loss: 0.0912 - val_accuracy: 0.9744
Epoch 23/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0600 - accuracy: 0.9818 - val_loss: 0.0910 - val_accuracy: 0.9755
Epoch 24/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0578 - accuracy: 0.9826 - val_loss: 0.0916 - val_accuracy: 0.9743
Epoch 25/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0559 - accuracy: 0.9829 - val_loss: 0.0835 - val_accuracy: 0.9772
Epoch 26/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0536 - accuracy: 0.9836 - val_loss: 0.0943 - val_accuracy: 0.9749
Epoch 27/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0515 - accuracy: 0.9846 - val_loss: 0.0916 - val_accuracy: 0.9748
Epoch 28/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0494 - accuracy: 0.9848 - val_loss: 0.0871 - val_accuracy: 0.9759
Epoch 29/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0485 - accuracy: 0.9854 - val_loss: 0.0918 - val_accuracy: 0.9756
Epoch 30/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0466 - accuracy: 0.9854 - val_loss: 0.0838 - val_accuracy: 0.9773
Epoch 31/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0454 - accuracy: 0.9861 - val_loss: 0.0861 - val_accuracy: 0.9773
Epoch 32/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0442 - accuracy: 0.9867 - val_loss: 0.0826 - val_accuracy: 0.9778
Epoch 33/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0427 - accuracy: 0.9872 - val_loss: 0.0918 - val_accuracy: 0.9770
Epoch 34/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0411 - accuracy: 0.9870 - val_loss: 0.0856 - val_accuracy: 0.9771
Epoch 35/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0398 - accuracy: 0.9879 - val_loss: 0.0797 - val_accuracy: 0.9784
Epoch 36/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0390 - accuracy: 0.9884 - val_loss: 0.0887 - val_accuracy: 0.9772
Epoch 37/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0387 - accuracy: 0.9888 - val_loss: 0.0851 - val_accuracy: 0.9781
Epoch 38/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0372 - accuracy: 0.9888 - val_loss: 0.0851 - val_accuracy: 0.9772
Epoch 39/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0367 - accuracy: 0.9890 - val_loss: 0.0822 - val_accuracy: 0.9790
Epoch 40/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0358 - accuracy: 0.9894 - val_loss: 0.0821 - val_accuracy: 0.9796
Epoch 41/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0348 - accuracy: 0.9895 - val_loss: 0.0849 - val_accuracy: 0.9772
Epoch 42/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0333 - accuracy: 0.9898 - val_loss: 0.0806 - val_accuracy: 0.9797
Epoch 43/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0326 - accuracy: 0.9900 - val_loss: 0.0883 - val_accuracy: 0.9778
Epoch 44/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0314 - accuracy: 0.9903 - val_loss: 0.0859 - val_accuracy: 0.9791
Epoch 45/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0305 - accuracy: 0.9907 - val_loss: 0.0876 - val_accuracy: 0.9772
Epoch 46/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0299 - accuracy: 0.9910 - val_loss: 0.0844 - val_accuracy: 0.9793
Epoch 47/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0295 - accuracy: 0.9909 - val_loss: 0.0779 - val_accuracy: 0.9807
Epoch 48/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0285 - accuracy: 0.9914 - val_loss: 0.0806 - val_accuracy: 0.9788
Epoch 49/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0284 - accuracy: 0.9912 - val_loss: 0.0800 - val_accuracy: 0.9793
Epoch 50/50
375/375 [==============================] - 2s 6ms/step - loss: 0.0273 - accuracy: 0.9916 - val_loss: 0.0870 - val_accuracy: 0.9786
```

In [6]:
```python
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')

acc_ax.plot(history.history['accuracy'], 'b', label='train acc')
acc_ax.plot(history.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()
```



(5) 모델 평가하기

In [7]:
```python
loss_and_accuracy = model.evaluate(X_test_norm, Y_test_onehot, batch_size=128, verbose=1)
print('loss : %.4f, accracy : %.4f'%(loss_and_accuracy[0], loss_and_accuracy[1]))
```

```
79/79 [==============================] - 0s 4ms/step - loss: 0.0777 - accuracy: 0.9805
loss : 0.0777, accracy : 0.9805
```

4.2 GAN with MNIST

In [8]:
```python
import os
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
```

In [9]:
```python
from keras.layers import Input, Dense, Dropout, LeakyReLU
from keras.models import Model, Sequential
from keras.datasets import mnist
from tensorflow.keras.optimizers import Adam
from keras import initializers
```

In [10]:
```python
# 실험을 재현하고 동일한 결과를 얻을 수 있는지 확인하기 위해 seed 를 설정합니다.
np.random.seed(10)

# 우리의 랜덤 노이즈 벡터의 차원을 설정합니다.
random_dim = 100
```

(1) 데이터셋

In [11]:
```python
def load_minst_data():
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
    x_train = (x_train.astype(np.float32) - 127.5)/127.5 # -1~1 사이의 값
    x_train = x_train.reshape(60000, 784)
    return (x_train, y_train, x_test, y_test) # Label에 대한 진행이 없어도 됨
```

(2-3) 모델링 / 모델 학습과정 설정

In [12]:
```python
# Adam Optimizer를 사용합니다.
def get_optimizer():
    return Adam(learning_rate=0.0002, beta_1=0.5)

# Generator 만들기
def get_generator(optimizer):
    generator = Sequential()
    generator.add(Dense(256, input_dim=random_dim, kernel_initializer=initializers.RandomNormal(stddev=0.02)))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(512))
```

```python
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(1024))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(784, activation='tanh'))
    generator.compile(loss='binary_crossentropy', optimizer=optimizer)
    return generator

# Discriminator 만들기
def get_discriminator(optimizer):
    discriminator = Sequential()
    discriminator.add(Dense(1024, input_dim=784, kernel_initializer=initializers.RandomNormal(stddev=0.02)))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(256))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(1, activation='sigmoid'))
    discriminator.compile(loss='binary_crossentropy', optimizer=optimizer)
    return discriminator
```

In [13]:

```python
def get_gan_network(discriminator, random_dim, generator, optimizer):
    # Coding Time
    discriminator.trainable = False # Generator와 Discriminator를 동시에 학습시 trainable을 False로 설정

    gan_input = Input(shape=(random_dim,)) # gan_input : 노이즈(100 차원)

    x = generator(gan_input) # X:이미지

    gan_output = discriminator(x) # gan_output : 이미지가 진짜인지 아닌지에 대한 확률

    gan = Model(inputs=gan_input, outputs=gan_output)
    gan.compile(loss='binary_crossentropy', optimizer=optimizer)
    return gan
```

In [14]:

```python
# 생성된 MNIST 이미지 출력
def plot_generated_images(epoch, generator, examples=100, dim=(10, 10), figsize=(10, 10)):
    noise = np.random.normal(0, 1, size=[examples, random_dim])
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(examples, 28, 28)

    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i], interpolation='nearest', cmap='gray_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('gan_generated_image_epoch_%d.png' % epoch)
```

(4) 모델 학습

In [15]:

```python
def train(epochs=1, batch_size=128):
    # train 데이터와 test 데이터를 가져옵니다.
    x_train, y_train, x_test, y_test = load_minst_data()

    # train 데이터를 128 사이즈의 batch 로 나눕니다.
    batch_count = x_train.shape[0] // batch_size

    # 우리의 GAN 네트워크를 만듭니다.
    adam = get_optimizer()
    generator = get_generator(adam)
    discriminator = get_discriminator(adam)
    gan = get_gan_network(discriminator, random_dim, generator, adam)

    for e in range(1, epochs+1):
        print('-'*15, 'Epoch %d' % e, '-'*15)
        for _ in tqdm(range(batch_count)):
            # Coding Time
            # 입력으로 사용할 random 노이즈와 이미지를 가져옵니다.
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            image_batch = x_train[np.random.randint(0, x_train.shape[0], size=batch_size)]

            # Generator를 통해 MNIST 이미지를 생성
            generated_images = generator.predict(noise, verbose =0)
            X = np.concatenate([image_batch, generated_images])

            # Discriminator 학습
            y_dis = np.zeros(2*batch_size)
            y_dis[:batch_size] = 0.9
            discriminator.trainable = True
            discriminator.train_on_batch(X, y_dis)

            # Generator 학습
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            y_gen = np.ones(batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y_gen)
        if e == 1 or e % 20 == 0:
            plot_generated_images(e, generator)
```

In [16]:

```python
train(20, 128)
```

```
--------------- Epoch 1 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.19it/s]
--------------- Epoch 2 ---------------
100%|██████████| 468/468 [00:28<00:00, 16.24it/s]
--------------- Epoch 3 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.57it/s]
--------------- Epoch 4 ---------------
100%|██████████| 468/468 [00:28<00:00, 16.35it/s]
--------------- Epoch 5 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.42it/s]
--------------- Epoch 6 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.39it/s]
--------------- Epoch 7 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.50it/s]
--------------- Epoch 8 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.35it/s]
--------------- Epoch 9 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.60it/s]
--------------- Epoch 10 ---------------
100%|██████████| 468/468 [00:31<00:00, 14.74it/s]
--------------- Epoch 11 ---------------
100%|██████████| 468/468 [00:31<00:00, 14.83it/s]
--------------- Epoch 12 ---------------
100%|██████████| 468/468 [00:25<00:00, 18.03it/s]
--------------- Epoch 13 ---------------
100%|██████████| 468/468 [00:31<00:00, 15.01it/s]
--------------- Epoch 14 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.53it/s]
--------------- Epoch 15 ---------------
100%|██████████| 468/468 [00:31<00:00, 14.71it/s]
--------------- Epoch 16 ---------------
100%|██████████| 468/468 [00:31<00:00, 14.66it/s]
--------------- Epoch 17 ---------------
100%|██████████| 468/468 [00:32<00:00, 14.37it/s]
--------------- Epoch 18 ---------------
100%|██████████| 468/468 [00:33<00:00, 13.86it/s]
--------------- Epoch 19 ---------------
100%|██████████| 468/468 [00:33<00:00, 14.07it/s]
--------------- Epoch 20 ---------------
100%|██████████| 468/468 [00:33<00:00, 13.94it/s]
```