6주차 실습과제

2017313107 이승태

1. 12장_CNNforMNIST

(1) imort와 model을 만들기 위한 변수를 넣어준다.

```
[1]  from __future__ import print_function
     import keras
     from keras.datasets import mnist
     from keras.models import Sequential
     from keras.layers import Dense,Dropout,Flatten
     from keras.layers import Conv2D, MaxPooling2D
     from keras import backend as K

[2]  batch_size = 128
     num_classes = 10
     epoch = 12
```

(2) mnist set을 불러온 다음 train과 test data로 나눈다. 그 후 categorical crossentropy를 사용하기 위해 to_categorical을 이용하여 바꾼다.

```
[3]  img_rows, img_cols = 28,28
     (x_train,y_train),(x_test,y_test) = mnist.load_data()
     if K.image_data_format() == 'channels_first':
       x_train = x_train.reshape(x_train.shape[0],1,img_rows,img_cols)
       x_test = x_test.reshape(x_test.shape[0],1,img_rows,img_cols)
       input_shape = (1,img_rows,img_cols)
     else:
       x_train = x_train.reshape(x_train.shape[0],img_rows,img_cols,1)
       x_test = x_test.reshape(x_test.shape[0],img_rows,img_cols,1)
       input_shape = (img_rows,img_cols,1)

[4]  x_train=x_train.astype('float32')/255
     x_test=x_test.astype('float32')/255

     %tensorflow_version 2.x
     import tensorflow as tf

     y_train = tf.keras.utils.to_categorical(y_train,num_classes)
     y_test = tf.keras.utils.to_categorical(y_test,num_classes)
```

(3) 모델을 다음과 같이 CNN을 이용해서 만들고, Adadelta를 이용하여 optimize했다.

```
[19]  model = Sequential()
      model.add(Conv2D(32, kernel_size=(3,3),
                       activation='relu',
                       input_shape=input_shape))
      model.add(MaxPooling2D(pool_size=(2,2)))
      model.add(Dropout(0.25))
      model.add(Flatten())
      model.add(Dense(128, activation='relu'))
      model.add(Dropout(0.5))
      model.add(Dense(num_classes, activation='softmax'))

[21]  model.compile(loss=keras.losses.categorical_crossentropy,
                    optimizer=tf.keras.optimizers.Adadelta(),
                    metrics=['accuracy'])
```

(4) 데이터를 이용해 model을 fit했다.

```
model.fit(x_train,y_train,
          batch_size=batch_size,
          epochs = epoch,
          verbose=1)
```

```
469/469 [==============================] - 2s 4ms/step - loss: 2.2343 - accuracy: 0.2111
Epoch 3/30
469/469 [==============================] - 2s 4ms/step - loss: 2.1706 - accuracy: 0.3107
Epoch 4/30
469/469 [==============================] - 2s 4ms/step - loss: 2.1012 - accuracy: 0.3897
Epoch 5/30
469/469 [==============================] - 2s 4ms/step - loss: 2.0312 - accuracy: 0.4487
Epoch 6/30
469/469 [==============================] - 2s 4ms/step - loss: 1.9578 - accuracy: 0.4892
Epoch 7/30
469/469 [==============================] - 2s 4ms/step - loss: 1.8816 - accuracy: 0.5196
Epoch 8/30
469/469 [==============================] - 2s 4ms/step - loss: 1.8064 - accuracy: 0.5431
Epoch 9/30
469/469 [==============================] - 2s 4ms/step - loss: 1.7286 - accuracy: 0.5690
Epoch 10/30
469/469 [==============================] - 2s 4ms/step - loss: 1.6514 - accuracy: 0.5867
Epoch 11/30
469/469 [==============================] - 2s 4ms/step - loss: 1.5780 - accuracy: 0.6045
Epoch 12/30
469/469 [==============================] - 2s 4ms/step - loss: 1.5107 - accuracy: 0.6201
Epoch 13/30
469/469 [==============================] - 2s 4ms/step - loss: 1.4440 - accuracy: 0.6342
Epoch 14/30
```

(5) test loss와 accuracy를 나타냈다.

```
score=model.evaluate(x_test,y_test,verbose=0)
print('Test loss : ',score[0])
print('Test acc : ',score[1])
```

```
Test loss :  0.6343984007835388
Test acc :  0.8600000143051147
```

(6) convolution 층을 조금 더 깊게 쌓아 모델을 만들고 성능을 평가해 보았다.
여러 층을 쌓는 것이 좀더 좋은 accuracy를 보였다.

```
[34]  model2 = Sequential()
      model2.add(Conv2D(32, kernel_size=(3,3),
                        activation='relu',
                        input_shape=input_shape))
      model2.add(Conv2D(32,(3,3),activation='relu'))
      model2.add(MaxPooling2D(pool_size=(2,2)))
      model2.add(Dropout(0.25))
      model2.add(Flatten())
      model2.add(Dense(128, activation='relu'))
      model2.add(Dropout(0.5))
      model2.add(Dense(num_classes, activation='softmax'))
```

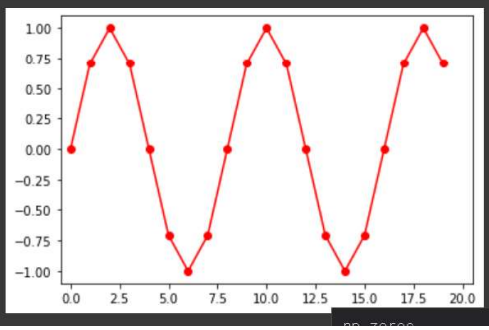
```
[35]  model2.compile(loss=keras.losses.categorical_crossentropy,
                     optimizer=tf.keras.optimizers.Adadelta(),
                     metrics=['accuracy'])
      model2.fit(x_train,y_train,
                 batch_size=batch_size,
                 epochs = epoch,
                 verbose=1)
      score=model2.evaluate(x_test,y_test,verbose=0)
      print('Test loss : ',score[0])
      print('Test acc : ',score[1])

      Epoch 1/30
      469/469 [==============================] - 3s 6ms/step - loss: 2.2995 - accuracy: 0.1127
      Epoch 2/30
      469/469 [==============================] - 3s 6ms/step - loss: 2.2565 - accuracy: 0.1689
      Epoch 3/30
      469/469 [==============================] - 3s 6ms/step - loss: 2.2093 - accuracy: 0.2264
      Epoch 4/30
      469/469 [==============================] - 3s 6ms/step - loss: 2.1531 - accuracy: 0.2885
      Epoch 5/30
      469/469 [==============================] - 3s 6ms/step - loss: 2.0823 - accuracy: 0.3519
      Epoch 6/30
      469/469 [==============================] - 3s 6ms/step - loss: 1.9972 - accuracy: 0.4067
      Epoch 7/30
      469/469 [==============================] - 3s 6ms/step - loss: 1.8994 - accuracy: 0.4507
      Epoch 8/30
      469/469 [==============================] - 3s 6ms/step - loss: 1.7905 - accuracy: 0.4943
      Epoch 9/30
      469/469 [==============================] - 3s 6ms/step - loss: 1.6691 - accuracy: 0.5353
```

```
Test loss :  0.4789749085903168
Test acc :  0.8726000189781189
```

2. 13장_RNN

(1) sin함수를 만들고 이를 데이터로 쓸 예정이다.

```
[1]  import numpy as np
     from matplotlib import pyplot as plt
     s = np.sin(2 * np.pi * 0.125 * np.arange(20))
     plt.plot(s,'ro-')
     plt.xlim(-0.5,20.5)
     plt.ylim(-1.1,1.1)
     plt.show()
```



(2) 위를 이용하여, train data를 만들었다.

```
[2]  from scipy.linalg import toeplitz
     S = np.fliplr(toeplitz(np.r_[s[-1], np.zeros(s.shape[0] - 2)], s[::-1]))
     S[:5, :3]
```

```
array([[ 0.00000000e+00,  7.07106781e-01,  1.00000000e+00],
       [ 7.07106781e-01,  1.00000000e+00,  7.07106781e-01],
       [ 1.00000000e+00,  7.07106781e-01,  1.22464680e-16],
       [ 7.07106781e-01,  1.22464680e-16, -7.07106781e-01],
       [ 1.22464680e-16, -7.07106781e-01, -1.00000000e+00]])
```

```
[3]  X_train = S[:-1, :3][:, :, np.newaxis]
     Y_train = S[:-1,3]
     X_train.shape, Y_train.shape
```

```
((18, 3, 1), (18,))
```

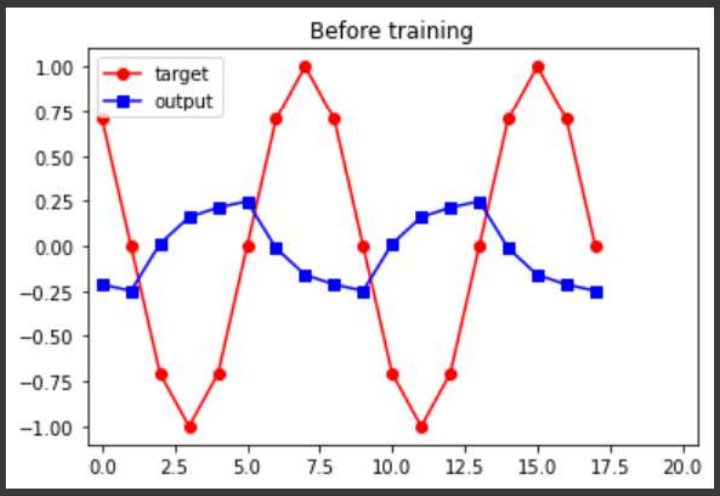
(3) RNN을 이용하여 모델을 만들었다.

```
[4]  from keras.models import Sequential
     from keras.layers import SimpleRNN, Dense

     np.random.seed(0)
     model = Sequential()
     model.add(SimpleRNN(10, input_shape=(3,1)))
     model.add(Dense(1, activation='linear'))
     model.compile(loss='mse', optimizer='sgd')
     model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 10)                120
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================
Total params: 131
Trainable params: 131
Non-trainable params: 0
_________________________________________________________________
```

(4) 처음 모델에 넣었을 때와 실제 값 비교

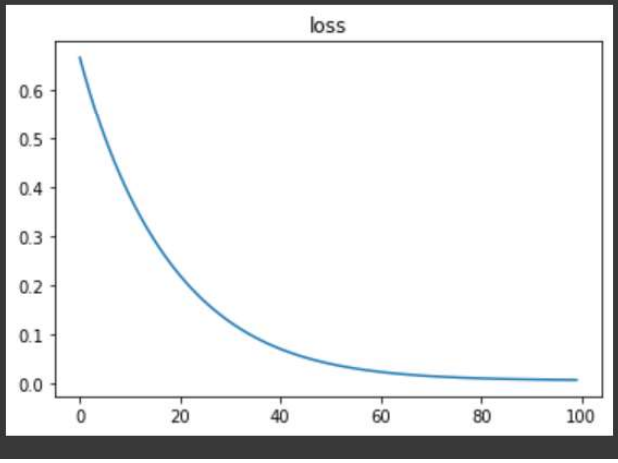
```
[5]  plt.plot(Y_train, 'ro-', label= 'target')
     plt.plot(model.predict(X_train[:,:,:]), 'bs-', label='output')
     plt.xlim(-0.5,20.5)
     plt.ylim(-1.1,1.1)
     plt.legend()
     plt.title('Before training')
     plt.show()
```



(5) model을 fit하고 loss 값을 그래프로 그려보았다.

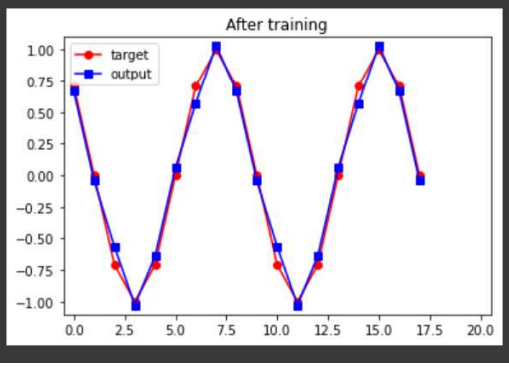
```
[6]  history = model.fit(X_train,Y_train, epochs = 100, verbose=0)
```

```
[7]  plt.plot(history.history['loss'])
     plt.title('loss')
     plt.show()
```



(6) 학습 후의 모습

```
[8]  plt.plot(Y_train, 'ro-', label= 'target')
     plt.plot(model.predict(X_train[:,:,:]), 'bs-', label='output')
     plt.xlim(-0.5,20.5)
     plt.ylim(-1.1,1.1)
     plt.legend()
     plt.title('After training')
     plt.show()
```



3. 13장_GRU+LSTM

(1) 위와 똑같은 방식으로 data를 만든다.

```
[1]  import numpy as np
     from matplotlib import pyplot as plt
     s = np.sin(2 * np.pi * 0.125 * np.arange(20))
     from scipy.linalg import toeplitz
     S = np.fliplr(toeplitz(np.r_[s[-1], np.zeros(s.shape[0] - 2)], s[::-1]))

     X_train = S[:-1, :3][:, :, np.newaxis]
     Y_train = S[:-1,3]
```

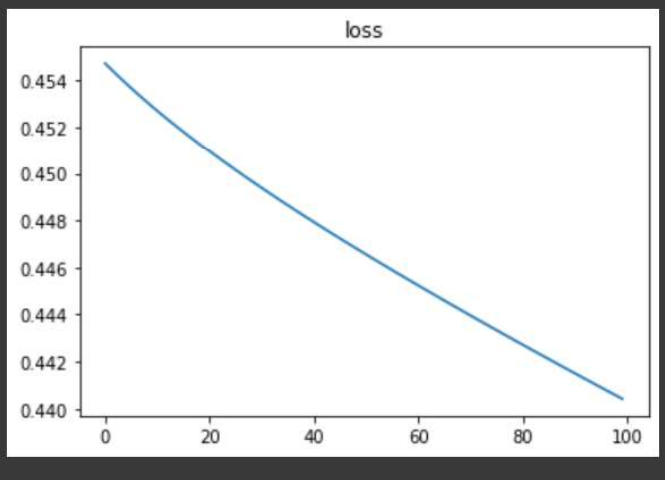
(2) 위는 simple rnn이었지만 lstm으로 바꾼다.

```
[2]  from keras.models import Sequential
     from keras.layers import LSTM, Dense

     np.random.seed(0)
     model = Sequential()
     model.add(LSTM(10, input_shape=(3,1)))
     model.add(Dense(1, activation='linear'))
     model.compile(loss='mse', optimizer='sgd')
     model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 10)                480
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================
Total params: 491
Trainable params: 491
Non-trainable params: 0
_________________________________________________________________
```

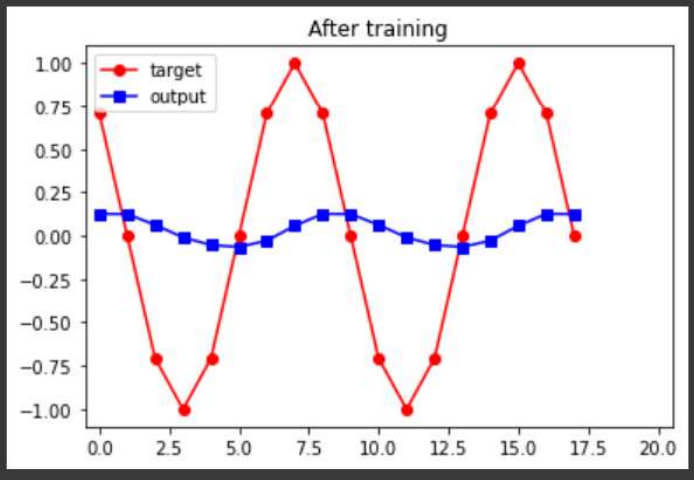
(3) 모델을 fit하자 loss값이 감소하였지만 위의 값보다 천천히 감소하였는데 이는 파라미터의 개수가 훨씬 더 많기 때문이라고 생각할 수 있다.

```
[3]  history = model.fit(X_train,Y_train, epochs = 100, verbose=0)
     plt.plot(history.history['loss'])
     plt.title('loss')
     plt.show()
```



(4) 그러므로 아직 완벽히 학습되지 않았다.

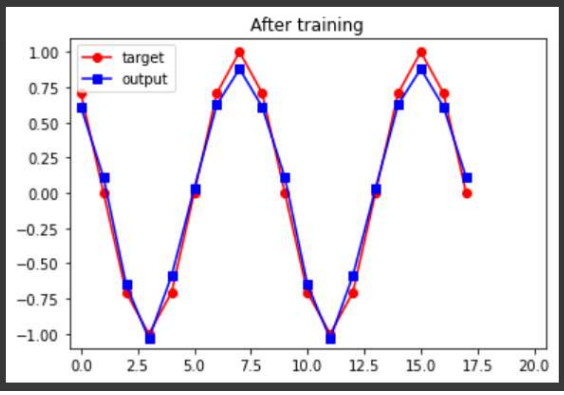
```
[4]  plt.plot(Y_train, 'ro-', label= 'target')
     plt.plot(model.predict(X_train[:,:,:]), 'bs-', label='output')
     plt.xlim(-0.5,20.5)
     plt.ylim(-1.1,1.1)
     plt.legend()
     plt.title('After training')
     plt.show()
```



(5) model의 epoch을 더 늘려 fit해보았다. 학습이 제대로 되었다.

```
[5]  history = model.fit(X_train,Y_train, epochs = 2000, verbose=0)
```

```
[6]  plt.plot(Y_train, 'ro-', label= 'target')
     plt.plot(model.predict(X_train[:,:,:]), 'bs-', label='output')
     plt.xlim(-0.5,20.5)
     plt.ylim(-1.1,1.1)
     plt.legend()
     plt.title('After training')
     plt.show()
```



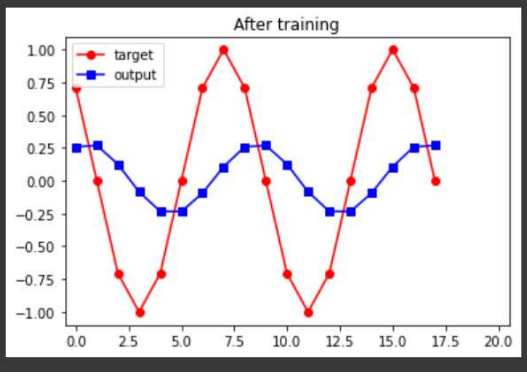
(6) 마찬가지로 GRU를 이용하여 모델을 생성해 내었다.

```
[7]  from keras.models import Sequential
     from keras.layers import GRU, Dense

     np.random.seed(0)
     model = Sequential()
     model.add(GRU(10, input_shape=(3,1)))
     model.add(Dense(1, activation='linear'))
     model.compile(loss='mse', optimizer='sgd')
     model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 10)                390
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 401
Trainable params: 401
Non-trainable params: 0
_________________________________________________________________
```

(7) LSTM과 마찬가지로 parameter가 많아 완벽하게 학습이 되지 않은 모양이다.

```
[8]  history = model.fit(X_train,Y_train, epochs = 100, verbose=0)

[9]  plt.plot(Y_train, 'ro-', label= 'target')
     plt.plot(model.predict(X_train[:,:,:]), 'bs-', label='output')
     plt.xlim(-0.5,20.5)
     plt.ylim(-1.1,1.1)
     plt.legend()
     plt.title('After training')
     plt.show()
```



4. 13장_RNNNforReuterdataset

(1) 기사 데이터를 load한 다음 test와 train data로 나누었다.

```
[1]  from tensorflow.keras.datasets import reuters

     %matplotlib inline
     import matplotlib.pyplot as plt
     import numpy as np
     (X_train, Y_train), (X_test, Y_test) = reuters.load_data(num_words=1000, test_split=0.2)

     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters.npz
     2113536/2110848 [==============================] - 0s 0us/step
     2121728/2110848 [==============================] - 0s 0us/step

[2]  print('Train data : {}'.format(len(X_train)))
     print('Test data : {}'.format(len(X_test)))
     num_classes = max(Y_train) + 1
     print('class : {}'.format(num_classes))

     Train data : 8982
     Test data : 2246
     class : 46

[3]  word_index = reuters.get_word_index()

     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters_word_index.json
     557056/550378 [==============================] - 0s 0us/step
     565248/550378 [==============================] - 0s 0us/step
```

(2) 새로운 import

```
[4]  index_word={}
     for key,value in word_index.items():
       index_word[value] = key

     from tensorflow.keras.models import Sequential , load_model
     from tensorflow.keras.layers import Dense, LSTM, Embedding
     from tensorflow.keras.preprocessing.sequence import pad_sequences
     from tensorflow.keras.utils import to_categorical
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

(3) train하고 test data를 나누고 Embedding layer를 이용하여 lstm을 구현해 주었다.

```
[5]  max_len = 100
     X_train = pad_sequences(X_train, maxlen=max_len)
     X_test = pad_sequences(X_test, maxlen=max_len)

     Y_train = to_categorical(Y_train)
     Y_test = to_categorical(Y_test)
```

```
[6]  model = Sequential()
     model.add(Embedding(1000,120))
     model.add(LSTM(120))
     model.add(Dense(46, activation='softmax'))
     es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose=1, patience=4)
     mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode = 'max', save_best_only =True)
     model.compile(loss= 'categorical_crossentropy', optimizer='adam', metrics = ['acc'])
```

(4) test accuracy를 측정

```
[8]  loaded = load_model('best_model.h5')
     print("₩n 테스트정확도 : %.4f" % (loaded.evaluate(X_test, Y_test)[1]))

     71/71 [==============================] - 2s 28ms/step - loss: 1.7123 - acc: 0.5637

      테스트정확도 : 0.5637
```

(5) model의 loss값을 epoch에 따라 그려보았다.

```
[9]  epochs = range(1, len(history.history['acc']) + 1)
     plt.plot(epochs, history.history['loss'])
     plt.plot(epochs, history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'val'], loc = 'upper left')
     plt.show()
```