

5주차 실습과제

2017313107 이승태

1. 10장_Colab_MNIST_MLP모형코딩

(1) 구글 드라이브 마운트 및 hello colab출력

```
[1] from google.colab import drive

drive.mount('/content/gdrive')

Mounted at /content/gdrive

[2] a='hello colab'
print(a)

hello colab
```

(2) train data와 test data를 drive에서 꺼내온다.

```
[4] #mnist_train(60000개)과 mnist_test(10000개) 데이터를 각각 불러온다.
data_file = open("/content/gdrive/My Drive/data/mnist_train.csv", "r") #연결되어 있는
training_data = data_file.readlines()
data_file.close()

test_data_file = open("/content/gdrive/My Drive/data/mnist_test.csv", "r")
test_data = test_data_file.readlines()
test_data_file.close()
```

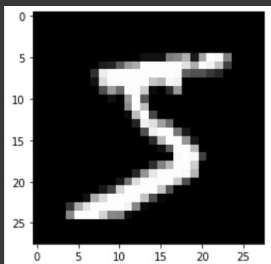
(3) 데이터를 가시적으로 표현

```
[5] #matplotlib과 numpy라이브러리를 불러온 후 데이터 하나를 시각화해본다.
import matplotlib.pyplot as plt
import numpy as np

t = np.asfarray(training_data[0].split(","))

# 일렬로 늘어선 픽셀정보를 28x28 행렬로 바꾼다
n = t[1:].reshape(28,28)

plt.imshow(n, cmap='gray')
plt.show()
```



(4) deep neural network 클래스를 정의하고 메소드들을 정의하였다.

predict, train, accuracy측정, sigmoid, normalize, tanh, softmax함수 등을 정의해 주었다.

```
[6] class DeepNeuralNetwork:
    #DeepNeuralNetwork 클래스를 initialize
    def __init__(self, input_layers, hidden_layer_1, hidden_layer_2, hidden_layer_3, output_layers):
        self.inputs = input_layers
        self.hidden_1 = hidden_layer_1
        self.hidden_2 = hidden_layer_2
        self.hidden_3 = hidden_layer_3
        self.outputs = output_layers
        self.test_data = None

        #가중치 값들을 모두 랜덤으로 초기화
        self.w_ih = np.random.randn(self.inputs, self.hidden_1) / np.sqrt(self.inputs/2)
        self.w_hh_12 = np.random.randn(self.hidden_1, self.hidden_2) / np.sqrt(self.hidden_1/2)
        self.w_hh_23 = np.random.randn(self.hidden_2, self.hidden_3) / np.sqrt(self.hidden_2/2)
        self.w_ho = np.random.randn(self.hidden_3, self.outputs) / np.sqrt(self.hidden_3/2)

    # feed-forward를 진행한다.
    def predict(self, x):
        # 문자열을 float array로 바꾸는 과정
        data = self.normalize(np.asfarray(x.split(',')))

        # 0번은 레이블이므로 제외
        data = data[1:]

        #3개의 은닉층(2개의 sigmoid와 1개의 tanh)과 하나의 출력층(softmax)
        layer_1 = self.sigmoid(np.dot(data, self.w_ih))
        layer_2 = self.tanh(np.dot(layer_1, self.w_hh_12))
        layer_3 = self.sigmoid(np.dot(layer_2, self.w_hh_23))
        output = self.softmax(np.dot(layer_3, self.w_ho))
        return output

    # training_data로 학습 진행
    def train(self, training_data, learning_rate, epoch):
        for ech in range(0, epoch):
            for i, x in enumerate(training_data):
                target = np.array(np.zeros(self.outputs) + learning_rate, ndmin=2)
                target[0][int(x[0])] = 1-learning_rate
                x = self.normalize(np.asfarray(x.split(',')))

                # feed-forward propagation
                layer1 = self.sigmoid(np.dot(x[1:], self.w_ih))
                layer2 = self.tanh(np.dot(layer1, self.w_hh_12))
                layer3 = self.sigmoid(np.dot(layer2, self.w_hh_23))
                layer4 = self.softmax(np.dot(layer3, self.w_ho))

                # back propagation
                layer4_reverse = (target - layer4)
                layer3_reverse = layer4_reverse.dot(self.w_ho.T) + (layer3 * (1 - layer3))
                layer2_reverse = layer3_reverse.dot(self.w_hh_23.T) + (1 - layer2) * (1 + layer2)
                layer1_reverse = layer2_reverse.dot(self.w_hh_12.T) + (layer1 * (1 - layer1))

                # weight update
                self.w_ho = self.w_ho + learning_rate * layer4_reverse.T.dot(np.array(layer3, ndmin=2)).T
                self.w_hh_23 = self.w_hh_23 + learning_rate * layer3_reverse.T.dot(np.array(layer2, ndmin=2)).T
                self.w_hh_12 = self.w_hh_12 + learning_rate * layer2_reverse.T.dot(np.array(layer1, ndmin=2)).T
                self.w_ih = self.w_ih + learning_rate * layer1_reverse.T.dot(np.array(x[1:], ndmin=2)).T

            #2000개에 한 번씩 accuracy 출력
            if i % 2000 == 0:
                self.print_accuracy()
```

```

# 현재 neural network의 accuracy를 출력한다.
def print_accuracy(self):
    matched = 0

    for x in self.test_data:
        label = int(x[0])
        predicted = np.argmax(self.predict(x))
        if label == predicted:
            matched = matched + 1
    print('accuracy : {0}'.format(matched/len(self.test_data)))

#sigmoid함수 정의
def sigmoid(self, x):
    return 1.0/(1.0 + np.exp(-x))

#feature scaling을 위한 normalize 함수 정의
def normalize(self, x):
    return (x / 255.0) * 0.99 + 0.01

#tanh함수 정의
def tanh(self, x):
    return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))

#softmax함수 정의
def softmax(self, x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

```

(5) 이를 이용하여, deep neural network를 만들고, train시켰다.

```

#input layer, hidden layer 1, 2, 3, output layer의 노드 수를 각각 784, 100, 100, 100, 10개로 설정
network = DeepNeuralNetwork(784, 100, 100, 100, 10)
network.test_data = test_data
#learning rate은 0.01, epoch는 1로 설정
network.train(training_data, 0.01, 1)

```

```

accuracy : 0.0892
accuracy : 0.5447
accuracy : 0.7799
accuracy : 0.8389
accuracy : 0.8541
accuracy : 0.8696
accuracy : 0.876
accuracy : 0.7999
accuracy : 0.8829
accuracy : 0.8781
accuracy : 0.8791
accuracy : 0.8781
accuracy : 0.8963
accuracy : 0.8869
accuracy : 0.8974
accuracy : 0.8662
accuracy : 0.9106
accuracy : 0.9145
accuracy : 0.8765
accuracy : 0.8941
accuracy : 0.9027
accuracy : 0.9035
accuracy : 0.911
accuracy : 0.9071
accuracy : 0.9149
accuracy : 0.9188
accuracy : 0.9028
accuracy : 0.9099
accuracy : 0.8911
accuracy : 0.9174

```

2. 11장_DNNforMNIST(+dropout)

(1) train와 test image들을 받고, label을 categorical_crossentropy를 이용하기 위하여, to_categorical을 이용하여 바꾸었다.

```
[2] from keras.datasets import mnist
    from keras import models
    from keras import layers
    from tensorflow.keras.utils import to_categorical

[3] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [=====] - 0s 0us/step
11501568/11490434 [=====] - 0s 0us/step

[4] train_images = train_images.reshape((60000, 28*28))
    train_images = train_images.astype('float32')/ 255
    test_images = test_images.reshape((10000, 28*28))
    test_images = test_images.astype('float32')/ 255

    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)
```

(2) dense layer만을 이용하여 모델을 생성하였다.

```
[5] model = models.Sequential()
    model.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

[6] model.summary()

Model: "sequential"
-----
Layer (type)                 Output Shape              Param #
-----
dense (Dense)                (None, 512)               401920
-----
dense_1 (Dense)              (None, 10)                 5130
-----
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
-----
```

(3) model을 fit하고, test accuracy를 측정했다.

```
[7] ##epoch수 변경으로 모델의 학습을 조절
    model.fit(train_images, train_labels, epochs=10, batch_size=128)

Epoch 1/10
469/469 [=====] - 18s 10ms/step - loss: 0.4370 - accuracy: 0.8718
Epoch 2/10
469/469 [=====] - 5s 10ms/step - loss: 0.1112 - accuracy: 0.9668
Epoch 3/10
469/469 [=====] - 5s 10ms/step - loss: 0.0693 - accuracy: 0.9795
Epoch 4/10
469/469 [=====] - 5s 10ms/step - loss: 0.0481 - accuracy: 0.9850
Epoch 5/10
469/469 [=====] - 5s 10ms/step - loss: 0.0374 - accuracy: 0.9884
Epoch 6/10
469/469 [=====] - 5s 10ms/step - loss: 0.0271 - accuracy: 0.9919
Epoch 7/10
469/469 [=====] - 5s 10ms/step - loss: 0.0203 - accuracy: 0.9942
Epoch 8/10
469/469 [=====] - 5s 10ms/step - loss: 0.0142 - accuracy: 0.9958
Epoch 9/10
469/469 [=====] - 5s 10ms/step - loss: 0.0111 - accuracy: 0.9969
Epoch 10/10
469/469 [=====] - 4s 10ms/step - loss: 0.0088 - accuracy: 0.9978
<keras.callbacks.History at 0x7f54c7525c50>

[8] test_loss, test_acc = model.evaluate(test_images, test_labels)
    test_acc

313/313 [=====] - 1s 2ms/step - loss: 0.0824 - accuracy: 0.9786
0.978600025177002
```

(4) dropout을 설정하고 model을 생성후 fit했다. 그 후, accuracy를 측정했을 때, train accuracy는 감소했지만, test accuracy를 비교했을 때, 증가한 것을 보아 overfitting을 방지할 수 있음을 알 수 있다.

```
[9] model2 = models.Sequential()
    model2.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
    model2.add(layers.Dropout(0.2, noise_shape=None, seed = None))
    model2.add(layers.Dense(10, activation='softmax'))
    model2.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

[10] model2.fit(train_images, train_labels, epochs=10, batch_size=128)

Epoch 1/10
469/469 [=====] - 6s 11ms/step - loss: 0.4506 - accuracy: 0.8689
Epoch 2/10
469/469 [=====] - 5s 10ms/step - loss: 0.1301 - accuracy: 0.9606
Epoch 3/10
469/469 [=====] - 5s 11ms/step - loss: 0.0835 - accuracy: 0.9746
Epoch 4/10
469/469 [=====] - 5s 11ms/step - loss: 0.0635 - accuracy: 0.9817
Epoch 5/10
469/469 [=====] - 5s 11ms/step - loss: 0.0504 - accuracy: 0.9846
Epoch 6/10
469/469 [=====] - 5s 10ms/step - loss: 0.0416 - accuracy: 0.9881
Epoch 7/10
469/469 [=====] - 5s 11ms/step - loss: 0.0348 - accuracy: 0.9895
Epoch 8/10
469/469 [=====] - 5s 11ms/step - loss: 0.0300 - accuracy: 0.9908
Epoch 9/10
469/469 [=====] - 5s 11ms/step - loss: 0.0249 - accuracy: 0.9924
Epoch 10/10
469/469 [=====] - 5s 11ms/step - loss: 0.0195 - accuracy: 0.9937
<keras.callbacks.History at 0x7f54c6a2b7d0>

[11] test_loss, test_acc = model2.evaluate(test_images, test_labels)
    test_acc

313/313 [=====] - 1s 2ms/step - loss: 0.0652 - accuracy: 0.9820
0.9819999933242798
```