[2024-fall]

인공지능개론

Project Report

대구경북과학기술원[DGIST]

융복합대학 기초학부

학번: 202011101

이름: 서정호

Project 1

Full (python) Code & Description

2024_MECH_Intro_AI > Project_1 > Project_1.py > ...

```python
### Project_1_code ###
print(f"StudentID: {202011101}\tName: {'서정호'}")

### import libraries ###
import numpy as np
from scipy.special import expit as sigmoid #
import matplotlib.pyplot as plt # library for plotting arrays, i.e, data visualization
import cv2 # OpenCV for image processing, b/c scipy.misc is deprecated #

"""
### numpy library: how to generate column vectors
## np.array.reshape(row, column): array to matrix
## np.array.ndim: dimension of array
# np.array(list, tuple, ...): 1D row vector
# np.array(list, tuple, ...).reshape(-1, 1): 2D column vector for matrix multiplication. Not same to np.array.T which is a 1D column vector.
# np.dot(matrix, matrix): matrix multiplication, including vector dot product
# np.zeros(row, column): matrix of zeros
# np.random.normal(mean, std, (row, column)): matrix of random numbers
# np.argmax(array): return index(int type)
# np.array(sequence, dtype=np.float64): array with float data type
"""

# As a result, every array is 2D column vector for matrix multiplication Furthermore, every array data type should be float.
# so I used np.array(sequence, dtype=np.float64).reshape(-1, 1) instead of np.array.T
"""
# training dataset: full-MNIST training dataset, 60,000
# testing dataset: my own hand-writing dataset(png, 28x28 pixels)
# performance: under 97% is okay.
"""
```

- 행렬 계산을 위한 numpy
- Activation function(sigmoid)을 위한 scipy.special
- Data visualization(graph)를 위한 matplotlib.pyplot
- Image processing을 위한 cv2(scipy.misd is deprecated) libraries를 사용하였다.
- 특히, array의 경우 행렬 곱 연산을 위해 모두 2D column vector로 처리하였다. 따라서 array로 변환 및 생성할 때, 일관적으로 reshape하였다. 이를 위하여 np.array(sequence, dtype=np.float64).reshape(-1, 1)를 주로 사용하였다.
- Training_data로 MNIST training dataset을, testing_data로 본 연구자가 직접 제작한 handwriting dataset을 사용한다.

```
30
31  ## MNIST dataset.csv ##
32  # (label, pixel_1, pixel_2, ..., pixel_784), 28*28 pixels = 784 pixels
33  # label: 0~9, pixel: 0~255 (0:black, 255:white) for brightness
34  # word: white, background: black: inversion needed (invert smaller hand-writing dataset which is word: black, background: white)
35  # each line: target + comma + pixel + comma + pixel +... + comma + pixel + '\n' => file.readlines(): list of str element(each line)
36
37  ## hand-writing dataset: 10*3 = 30 ##
38  # 1_Tablet_capture_image
39  # 2_hand_writing_image_phone_scan
40  # 3_MS_word_capture_image ##
41
42  ## hyper-parameters values ##
43  input_size = 784 # 28*28
44  hidden_size = 200 # experimentally determined
45  output_size = 10 # 0~9
46  learning_rate = [0.1, 0.2] # experimentally determined
47  epochs = [1, 2, 4, 5, 6, 7, 10, 20] # experimentally determined
48
49
50  ## loading training dataset ##
51  with open("../MNIST_dataset_csv/60,000_full_mnist_train.csv", 'r') as f:
52      MNIST_training_data_list = f.readlines() # list of str element(each element)
53
54  ## loading testing dataset: 30 hand-writing data ##
55  # (label, path)
56  hand1_path_list = [ (label, f"./handwriting_images/1_Tablet_capture_image/1_{label}.png") for label in range(0,10) ] # label: 0~9
57  hand2_path_list = [ (label, f"./handwriting_images/2_hand_writing_image_phone_scan/2_{label}.png") for label in range(0,10) ] # label: 0~9
58  hand3_path_list = [ (label, f"./handwriting_images/3_MS_word_capture_image/3_{label}.png") for label in range(0,10) ] # label: 0~9
59
```

- 성능을 측정하기 위한 hyper-parameters를 사전에 initialization하였다.
- 상대경로를 사용하여 사전에 local 환경에 저장해둔 MNIST dataset, handwriting dataset을 loading하였다.

```
60
61  ### Preprocessing for MNIST data ###
62
63  ## re-scale(normalization, 0.0 to 1.0 & shift, 0.01 to 1.00)
64  def re_scale(img_array):
65      return (img_array / 255.0 * 0.99) + 0.01
66
67  ## one-hot encoding for label ##
68  def encoding(label):
69      label_array = np.zeros(output_size).reshape(-1,1) + 0.01
70      label_array[int(label)] = 0.99
71      return label_array
72
73  ## training_data Preprocessing ##
74  training_data = []
75  for element in MNIST_training_data_list: # str
76      # split the element by the ',' commas
77      all_values = element.split(',') # str to list, all_values: [ "label", "pixel_1", "pixel_2", ..., "pixel_784" ]
78
79      # Preprocessing
80      img_array = re_scale( np.array(all_values[1:], dtype=np.float64).reshape(-1,1) )
81      label_array = encoding(all_values[0])
82
83      training_data += [ (img_array, label_array) ] # (img_array, label_array) of float
84
```

- MNIST data를 preprocessing하기 위한 re-scale, encoding 함수를 정의하였다. Encoding 함수는 주어진 label을 one-hot encoding을 통해 array로 만드는 기행을 수행한다.
- 이후 주어진 함수를 사용하여 loading한 MNIST training dataset을 preprocessing하였다.

```
### Preprocessing for Hand-writing data ###

## image preprocessing: image to array ##
def path_to_array(path, target_size = (28,28)):
    # read the image as grayscale
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

    # resize the image
    img_resized = cv2.resize(img, target_size) # 28*28 = 784 pixels

    # color binarization: words are black(0), background is white(255)
    _, img_bin = cv2.threshold(img_resized, 128, 255, cv2.THRESH_BINARY) # _ = threshold 128

    # # convert to 1D array
    # img_array = img_bin.flatten()
    img_array = img_bin.reshape(-1,1)
    return img_array

## inversion for handwriting_data ##
def invert(img_array):
    return 255.0 - img_array


testing_data = [ ( re_scale(invert(path_to_array(path))), encoding(label) ) for label, path in hand1_path_list + hand2_path_list +
hand3_path_list ] # (img_array, label_array) of float

```

- Loading한 handwriting data를 preprocessing하기 위해 path_to_array, invert 함수를 정의하였다. handwriting data의 경우 png파일로 local에 저장되어, 위에서는 path만을 loading하였기에, 이를 img_array로 반환하는 함수가 필요하다. 또한 black, white 두 값으로 pixels을 color binarization하였다.
- 또한, MNIST의 경우 배경이 검은색, 글씨가 흰색이다. 그러나 handwriting data의 경우 배경이 흰색, 글씨가 검은색이므로, invert 함수를 통해 inversion하였다.
- 마지막으로, 주어진 함수를 사용하여 testing_data로 만들었다.

```
## Neural Network Class Definition ##
class Neural_Network: # MLP
    # Initialize the neural network: Constructor #
    def __init__(self, learning_rate, input_size = input_size, hidden_size = hidden_size, output_size = output_size):
        # set number of nodes in each input, hidden, output layer
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # link Weights matrices, W_ih and W_ho
        # Initialize Weights matrix with gaussian distribution
        self.W_ih = np.random.normal(0.0, pow(self.hidden_size, -0.5), (self.hidden_size, self.input_size))
        self.W_ho = np.random.normal(0.0, pow(self.output_size, -0.5), (self.output_size, self.hidden_size))

        # learning rate
        self.lr = learning_rate

        # activation function: sigmoid function
        self.activation_function = lambda x: sigmoid(x) # def, return omission is okay in lambda function, generally used for simple functions

```

- Classification model의 class definition이다. Instance가 호출될 때마다 Neural Network를 initialization하기 위해, constructor __init__를 사용하여 input_size, hidden_size, output_size를 default value로 설정하였다. (# of nodes in each layers)
- 또한 Weights matrix를 gaussian distribution을 따르도록 initialization하고, instance의 learning은 추후 입력되는 값을 사용한다.
- Activation function은 sigmoid function을 사용하였다.

```
    # train the neural network #
    def train(self, input_array, target_array):
        # convert list to matrix: [1, 2, 3, 4] to [[1], [2], [3], [4]].T
        inputs = input_array
        targets = target_array

        ## Forward Propagation ##

        # input to hidden layer
        hidden_inputs = np.dot(self.W_ih, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)

        # hidden to output layer
        final_inputs = np.dot(self.W_ho, hidden_outputs)
        final_outputs = self.activation_function(final_inputs)

        ## Backward Propagation ##
        # error
        output_errors = targets - final_outputs

        # hidden_errors are distributed according to the proportion of weights from output_errors
        # ignore the denominator for simplification
        hidden_errors = np.dot(self.W_ho.T, output_errors)

        # update the weights
        self.W_ho += self.lr * np.dot((output_errors * final_outputs * (1.0 - final_outputs)), np.transpose(hidden_outputs))
        self.W_ih += self.lr * np.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), np.transpose(inputs))
```

- training과정을 위한 class의 member function이다. 각 2D column vector을 parameters로 가지고, 각 arguments는 위의 Weights matrix와 행렬 곱 연산이 이루어진다. 또한 backward propagation을 통해 error가 역전파되고, 이는 기존의 weights를 update한다.

```
    # query the neural network #
    def query(self, img_array):
        inputs = img_array

        ## Forward Propagation ##

        # input to hidden layer
        hidden_inputs = np.dot(self.W_ih, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)

        # hidden to output layer
        final_inputs = np.dot(self.W_ho, hidden_outputs)
        final_outputs = self.activation_function(final_inputs)

        return final_outputs
```

- model에 new query의 input(img_array)이 들어오면, forward propagation을 통해 output을 반환하는 class의 member function이다.

```
176
177  # Initialize for the performance graph #
178  pts = { lr : [ ] for lr in learning_rate } # value: (epoch, accuracy)
179
180  ## train & test the neural network ##
181  for lr in learning_rate: # 0.1, 0.2
182      for epoch in epochs: # 1, 2, 4, 5, 6, 7, 10, 20
183          # initialization new instance for each epoch
184          instance = Neural_Network(lr) # learning_rate, input_size = 784, hidden_size = 200, output_size = 10
185          ### train the neural network ###
186          for _ in range(epoch):
187              for img_array, label_array in training_data:
188                  instance.train(img_array, label_array) # present instance's weights update
```

- epoch-performance graph를 그리기 위해, each learning_rate(lr)에 대한 (epoch, accuracy) tuple이 담긴 list를 pts dictionary의 value로 initialization하였다.
- 이후, each learning_rate에서 epoch에 대한 performance를 계산해야 한다. 따라서 each epoch에 대해 새로운 instance(NN model)을 initialization하고, training_data를 training하였다. 이 때 Weights가 update되었다.

```
190          ### test the neural network ###
191          # scorecard for how well the network performs, initially empty
192          scorecard = []
193          for img_array, label_array in testing_data: # (img_array, label_array) of float
194              output_array = instance.query(img_array) # using updated weights
195
196              # the index of the highest value corresponds to the label
197              output = np.argmax(output_array)
198              target = np.argmax(label_array)
199              # append correct or incorrect to list
200              if (output == target):
201                  # network's answer matches correct answer, add 1 to scorecard
202                  scorecard.append(1)
203              else:
204                  # network's answer doesn't match correct answer, add 0 to scorecard
205                  scorecard.append(0)
206
207          # calculate the performance score, the fraction of correct answers
208          scorecard_array = np.array(scorecard, dtype=np.float64)
209          # performance score
210          accuracy = scorecard_array.sum() / scorecard_array.size
211          # print accuracy
212          print(f"Performance: {accuracy} | epoch: {epoch}, learning rate: {lr}")
213          # store coordinate data for result graph #
214          pts[lr].append( (epoch, accuracy) )
215
```

- Updated weights로 performance를 측정하기 위해, testing_data를 통해 testing을 진행하였다. 이 때 performance를 계산하기 위한 idea로, scorecard list를 사용하였다. Output과 target이 같다면 1을, 다르다면 0을 추가한다. 따라서 1의 개수는 성공 횟수이고, 모든 개수는 전체 횟수이다. 0은 덧셈에 영향을 주지 않으므로, 1의 개수가 곧 모든 element의 합을 의미한다.

- 이후, epoch, learning rate에 따른 Performance를 출력하고, 추후 graph를 그리기 위해 (epoch, accuracy)를 사전에 만들어둔 pts의 each learning rate의 value에 저장한다.

```
216
217  ## visualizing the results with graph: including marker and label ##
218  x = { lr : [ epoch for epoch, accuracy in pts[lr] ] for lr in learning_rate }
219  y = { lr : [ accuracy for epoch, accuracy in pts[lr] ] for lr in learning_rate }
220
221  ## plot the graph together ##
222  plt.plot( x[learning_rate[0]], y[learning_rate[0]], 'rD-', label = f'lr = {learning_rate[0]}' ) # lr = 0.1
223  plt.plot( x[learning_rate[1]], y[learning_rate[1]], 'bs-', label = f'lr = {learning_rate[1]}' ) # lr = 0.1
224
225  # title of the graph
226  plt.title("Performance and Epoch\nMNIST dataset with 3-layer neural network")
227  # title of each axis
228  plt.xlabel("number of epochs")
229  plt.ylabel("performance")
230  # label of each graph
231  plt.legend()
232  # grid of y-axis
233  plt.grid(axis = 'y')
234
235  plt.show()
```

- Graph를 그리기 위한 코드이다. Each learning_rate에 대해 epoch값만을 모은 list를 x-axis에 출력하고, accuracy값만을 모은 list는 y-axis에 출력될 것이다. 그 외 graph에서 각 점의 모양과 색깔, 제목 등은 적절히 강의 slide와 유사하게 제작하였다.

Print_Result

```
C:\Users\HOME\AppData\Local\Programs\Python\Python313\python.exe C:\Users\HOME\Desktop\VS_Code\2024_MECH_Intro_AI\Project_1\Project_1.py
StudentID: 202011101    Name: 서정호
Performance: 0.8333333333333334 | epoch: 1, learning rate: 0.1
Performance: 0.8333333333333334 | epoch: 2, learning rate: 0.1
Performance: 0.8333333333333334 | epoch: 4, learning rate: 0.1
Performance: 0.8333333333333334 | epoch: 5, learning rate: 0.1
Performance: 0.7333333333333333 | epoch: 6, learning rate: 0.1
Performance: 0.8 | epoch: 7, learning rate: 0.1
Performance: 0.9333333333333333 | epoch: 10, learning rate: 0.1
Performance: 0.7333333333333333 | epoch: 20, learning rate: 0.1
Performance: 0.8 | epoch: 1, learning rate: 0.2
Performance: 0.8333333333333334 | epoch: 2, learning rate: 0.2
Performance: 0.8333333333333334 | epoch: 4, learning rate: 0.2
Performance: 0.8 | epoch: 5, learning rate: 0.2
Performance: 0.8666666666666667 | epoch: 6, learning rate: 0.2
Performance: 0.8 | epoch: 7, learning rate: 0.2
Performance: 0.7666666666666667 | epoch: 10, learning rate: 0.2
Performance: 0.8 | epoch: 20, learning rate: 0.2

Process finished with exit code 0
```

StudentID: 202011101 Name: 서정호

Performance: 0.8333333333333334 | epoch: 1, learning rate: 0.1

Performance: 0.8333333333333334 | epoch: 2, learning rate: 0.1

Performance: 0.8333333333333334 | epoch: 4, learning rate: 0.1

Performance: 0.8333333333333334 | epoch: 5, learning rate: 0.1

Performance: 0.7333333333333333 | epoch: 6, learning rate: 0.1

Performance: 0.8 | epoch: 7, learning rate: 0.1

Performance: 0.9333333333333333 | epoch: 10, learning rate: 0.1

Performance: 0.7333333333333333 | epoch: 20, learning rate: 0.1

Performance: 0.8 | epoch: 1, learning rate: 0.2

Performance: 0.8333333333333334 | epoch: 2, learning rate: 0.2

Performance: 0.8333333333333334 | epoch: 4, learning rate: 0.2

Performance: 0.8 | epoch: 5, learning rate: 0.2

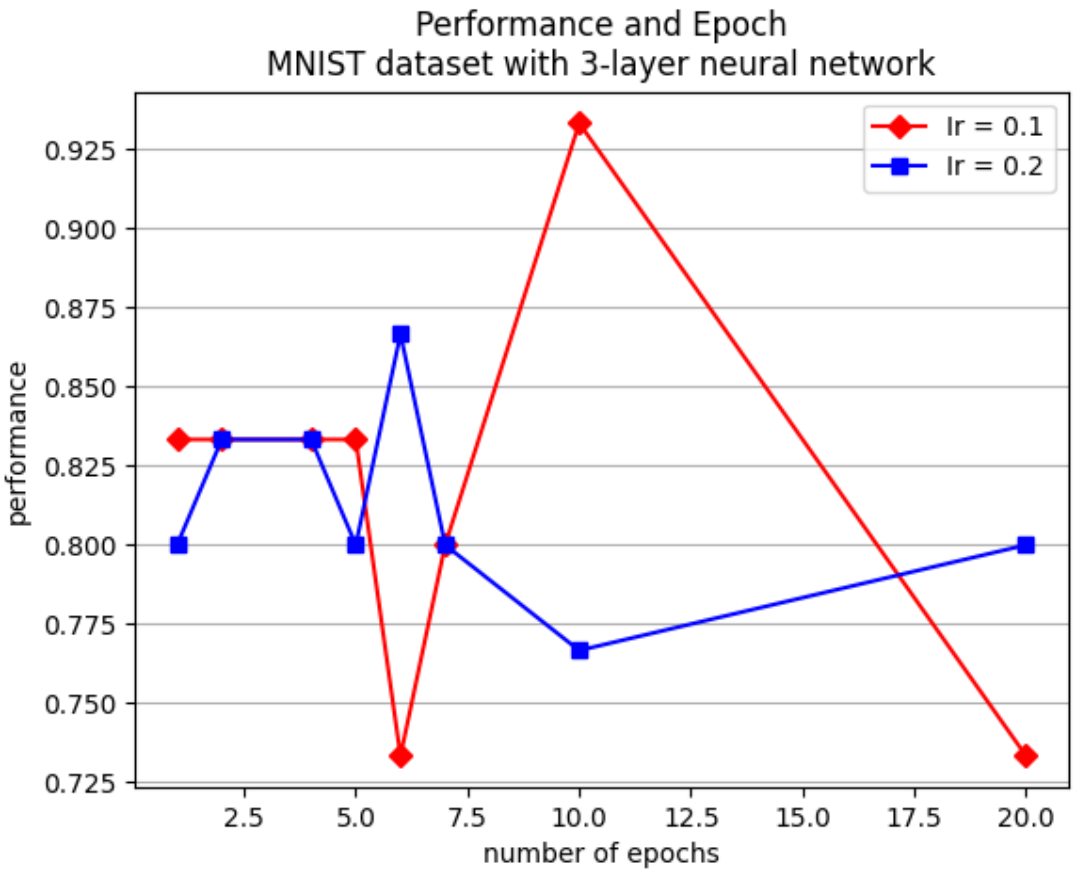
Performance: 0.8666666666666667 | epoch: 6, learning rate: 0.2

Performance: 0.8 | epoch: 7, learning rate: 0.2

Performance: 0.7666666666666667 | epoch: 10, learning rate: 0.2

Performance: 0.8 | epoch: 20, learning rate: 0.2

Graph



- Testing에 사용한 handwriting dataset은 아래와 같이 제작되어 classification model 성능 측정에 활용되었다.
  # 1_Tablet_capture_image: 본 연구자가 tablet에 작성한 글씨를 캡쳐한 이미지.
  # 2_hand_writing_image_phone_scan: 본 연구자가 종이에 작성한 글씨를 핸드폰으로 스캔한 이미지.
  # 3_MS_word_capture_image: 본 연구자가 word에 입력한 글씨를 캡쳐한 이미지.
- lr = 0.1의 경우, epoch에 따라 성능이 급격하게 변함을 알 수 있다. overfitting문제일 가능성이 있다. lr = 0.2의 경우, 성능 변화가 비교적 완만하므로, 학습이 잘 진행되었음을 알 수 있다.
- Performance: 0.9333333333333333 | epoch: 10, learning rate: 0.1일 때 가장 높게 나타났는데, learning rate: 0.1로 진행한 Project 2의 Performance: 0.9772 | angle: 0, epoch: 10에서 확인한 값보다 성능이 낮음을 알 수 있다. 둘 모두 training_data로 MNIST training dataset을 사용하였고, testing_data는 각각 handwriting dataset, MNIST testing dataset로 상이하였다.
  Project 2에서 angle: 0은 data augmentation을 하지 않은 training_data이다.

- Testing_data의 차이로 인해 발생한 **classification model의 성능 저하의 원인**은 training dataset과 testing dataset의 **distribution shift**가 유력하다. 즉 다시 말하자면, training에 사용한 MNIST dataset과 testing에 사용한 handwriting dataset의 distribution이 다를 것이다. Distribution 차이를 분석하면 아래와 같다.

  (1) MNIST dataset의 경우 **0-9까지의 label 샘플의 개수**가 균등하지 않으나, handwriting dataset의 경우 각 label 샘플의 개수를 3개씩 균등하게 제작하였다.

  (2) MNIST dataset의 경우 고등학생과 미국 인구조사국 직원의 숫자 샘플을 스캔하여 디지털화되었다. 그러나 handwriting dataset의 경우, 본 연구자가 작성 및 스캔한 샘플(# 2)에 더하여 디지털 글씨 캡쳐(# 1)와 디지털 문서 작성 프로그램 글씨 캡쳐(# 3)와 같이 다양한 tool을 사용하여 데이터를 제작하였다.

  따라서 MNIST dataset과 handwriting dataset 모두 특정 연령대와 교육 수준(젊고 일정 수준 이상의 교육을 받은)을 지닌 피실험자의, 규칙적이며 비교적 단순한 필체일 가능성이 크다. (특히, MS word에 입력된 글씨는 강력하게 규칙적이다.) 즉, 모든 dataset에서 **bias가 존재**한다.

  (3) 이 때 MNIST training dataset과 MNIST testing dataset은 대부분 미국인이 작성한 샘플이지만, MNIST training dataset과 handwriting dataset은 각각 미국인과 한국인이 작성한 샘플이므로 두 distribution을 더욱 상이하게 만든다. 미국인과 한국인은 인종과 문화 등의 차이로 인해 다른 필체 스타일을 가질 가능성이 크기 때문이다. **(Domain Gap)**
- 또 다른 성능 저하의 원인은 MNIST dataset과 handwriting dataset의 **이미지 전처리 방식(image preprocessing)의 차이**이다. Handwriting dataset의 경우 MNIST와 유사하게 배경과 글씨 색을 조정하고 정규화를 하였으나, 그 외 요인의 경우 상이할 수 있다.
- 따라서 **handwriting dataset으로 testing하였을 때 성능을 더욱 높일 수 있는 방안**이 필요하다. 이는 model 자체를 개선(MLP에서 CNN으로 변경, hidden layer size 최적화, hidden layer 개수 증가, activation function을 ReLU로 변경, loss function을 Cross-Entropy로 변경, learning rate 조정 등)할 수도 있으나, **dataset을 조정하는 방식**으로도 충분히 가능하다.
- (**Image preprocessing adjustment**) MNIST dataset에서의 image preprocessing과 같은 방식으로 handwriting dataset의 image preprocessing을 보완함으로써, **image preprocessing** 문제를 해결할 수 있다.
- (**Mixed Dataset Training**) MNIST training dataset에 handwriting dataset 일부를 추가하여 model을 학습시킴으로써, **distribution shift** 문제를 해결할 수 있다. 단, testing을 진행할 때에는 training_data에 추가한 handwriting data를 testing_data에서 제거함으로써 중복을 피해야 한다.

- (**Data Augmentation**) 모델이 학습할 data의 양을 증강함으로써 모델의 성능을 개선할 수 있다. 따라서 후술할 Project 2에서는 MNIST training dataset을 rotation을 통해 3배로 증강함으로써 모델의 성능을 개선한다.

Project 2

Full (python) Code & Description

```python
### Project_2_code ###
print(f"StudentID: {202011101}\tName: {'서정호'}")
print("angle = 0: before training dataset augmentation for comparison.")

### import libraries ###
import numpy as np
from scipy.special import expit as sigmoid #
import matplotlib.pyplot as plt # library for plotting arrays, i.e, data visualization
import cv2 # OpenCV for image processing, b/c scipy.misc is deprecated #
from scipy.ndimage import rotate # scipy.ndimage for image rotation, interpolation.rotate was changed to rotate.

"""
### numpy library: how to generate column vectors
## np.array.reshape(row, column): array to matrix
## np.array.ndim: dimension of array
# np.array(list, tuple, ...): 1D row vector
# np.array(list, tuple, ...).reshape(-1, 1): 2D column vector for matrix multiplication. Not same to np.array.T which is a 1D column vector.
# np.dot(matrix, matrix): matrix multiplication, including vector dot product
# np.zeros(row, column): matrix of zeros
# np.random.normal(mean, std, (row, column)): matrix of random numbers
# np.argmax(array): return index(int type)
# np.array(sequence, dtype=np.float64): array with float data type
"""

# As a result, every array is 2D column vector for matrix multiplication Furthermore, every array data type should be float.
# so I used np.array(sequence, dtype=np.float64).reshape(-1, 1) instead of np.array.T

"""
# training dataset: augmentation full-MNIST training dataset with rotation(5, 10, 15, 20, 25), 60,000 * 3 = 180,000
# testing dataset: full-MNIST testing dataset, 10,000
# performance: check to 98%.
"""
```

- Project1의 code와 유사하다. Rotation을 통해 data augmentation을 진행하므로, rotation을 위한 scipy.ndimage library를 추가하였다.
- Training_data로 MNIST training dataset을, testing_data로 MNIST testing dataset을 사용하였다. 이 때, MNIST training dataset을 augmentation을 통해 성능 향상을 유도하였다.

```python

## MNIST dataset.csv ##
# (label, pixel_1, pixel_2, ..., pixel_784), 28*28 pixels = 784 pixels
# label: 0~9, pixel: 0~255 (0:black, 255:white) for brightness
# word: white, background: black: inversion needed (invert smaller hand-writing dataset which is word: black, background: white)
# each line: target + comma + pixel + comma + pixel +... + comma + pixel + '\n' => file.readlines(): list of str element(each line)

## hyper-parameters values ##
input_size = 784 # 28*28
hidden_size = 200 # experimentally determined
output_size = 10 # 0~9
learning_rate = 0.1 # experimentally determined
epochs = [5, 10]

## rotation angles for augmentation ##
angles = [0, 5, 10, 15, 20, 25] # angle = 0: Before training dataset augmentation for comparison.


## loading training dataset ##
with open("../MNIST_dataset_csv/60,000_full_mnist_train.csv", 'r') as f:
    MNIST_training_data_list = f.readlines() # list of str element(each element)

## loading testing dataset ##
with open("../MNIST_dataset_csv/10,000_full_mnist_test.csv", 'r') as f:
    MNIST_testing_data_list = f.readlines() # list of str element(each element)

```

- 마찬가지로 hyper-parameters를 initialization하고, augmentation을 위한 rotation의 angle를 세팅하였다. 이 때, angle = 0은 augmentation을 하지 않은, MNIST training dataset을 의미한다. Augmentation을 통한 성능 향상을 확인하기 위해 값을 설정하였다.
- 이후 project 1과 마찬가지로 data를 loading하였다.

```python

### Preprocessing for MNIST data ###

## re-scale(normalization, 0.0 to 1.0 & shift, 0.01 to 1.00)
def re_scale(img_array):
    return (img_array / 255.0 * 0.99) + 0.01

## one-hot encoding for label ##
def encoding(label):
    label_array = np.zeros(output_size).reshape(-1,1) + 0.01
    label_array[int(label)] = 0.99
    return label_array

def data_preprocessing(data_list):
    data = []
    for element in data_list: # str
        # split each element by the ',' commas
        all_values = element.split(',') # str to list, all_values: [ "label", "pixel_1", "pixel_2", ..., "pixel_784" ]

        # Preprocessing
        img_array = re_scale( np.array(all_values[1:], dtype=np.float64).reshape(-1,1) )
        label_array = encoding(all_values[0])

        data += [ (img_array, label_array) ] # (img_array, label_array) of float

    return data

training_data = data_preprocessing(MNIST_training_data_list) # (img_array, label_array) of float
testing_data = data_preprocessing(MNIST_testing_data_list) # (img_array, label_array) of float

```

- MNIST data preprocessing을 위한 re_scale, encoding 함수를 만들어 사용하였다. 이 때, data_preprocessing 함수는 loading한 data를 array로 변환하여 반환한다.

```
## data_augmentation: rotation ##
def rotate_image(img_array, label_array, angle): # img_array: 2D column vector, 784x1
    if angle == 0:
        return img_array, label_array
    else:
        ## parameters of rotate function ##
        # input array: should be at least 2D, thus reshape to (28,28) matrix is better than 2D column vector(sometimes cause error)
        # (optional)cval: padding value, default 0.0
        # (optional)reshape: output shape, default True

        # reshape to 28x28 image
        img_matrix= img_array.reshape(28, 28)
        anticlockwise_matrix = rotate(img_matrix, angle, cval=0.01, reshape=False)  # angle: positive
        clockwise_matrix = rotate(img_matrix, -angle, cval=0.01, reshape=False)

        # reshape to 2D column vector
        original_array = img_array
        anticlockwise_array = anticlockwise_matrix.reshape(-1,1)
        clockwise_array = clockwise_matrix.reshape(-1,1)
        return (img_array, label_array), (anticlockwise_array, label_array), (clockwise_array, label_array) # tuple of 3 (img_array, label_array)


## training_data update: Augmentation ##
# (dict) key: angle, value: list of (img_array, label_array)
augmented_training_data = dict() # initialization dictionary
augmented_training_data.update( { 0 : training_data } )
augmented_training_data.update(
    { angle : [ rotate_image(img_array, label_array, angle)[0] for img_array, label_array in training_data ]
    + [ rotate_image(img_array, label_array, angle)[1] for img_array, label_array in training_data ]
    + [ rotate_image(img_array, label_array, angle)[2] for img_array, label_array in training_data ]
    for angle in angles if angle != 0 }
    )

```

- Training_data의 data augmentation을 위해 rotate_image 함수를 정의하였다. 이 때 rotate함수가 square image matrix만 인자로 삼아 반환하므로, reshape를 통해 data를 처리하였다. 이 때, **rotation을 한 이후 빈 여백은 padding value인 cval로 채워지게 된다.**
- 이후 each angle에 대한 augmented_training_data를 dictionary를 사용하여 정리하였다.

```
## Neural Network Class Definition ##
class Neural_Network: # MLP
    # Initialize the neural network: Constructor #
    def __init__(self, input_size = input_size, hidden_size = hidden_size, output_size = output_size, learning_rate = learning_rate):
        # set number of nodes in each input, hidden, output layer
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # link Weights matrices, W_ih and W_ho
        # Initialize Weights matrix with gaussian distribution
        self.W_ih = np.random.normal(0.0, pow(self.hidden_size, -0.5), (self.hidden_size, self.input_size))
        self.W_ho = np.random.normal(0.0, pow(self.output_size, -0.5), (self.output_size, self.hidden_size))

        # learning rate
        self.lr = learning_rate

        # activation function: sigmoid function
        self.activation_function = lambda x: sigmoid(x) # def, return omission is okay in lambda function, generally used for simple functions

```

- Neural_Network class definition이다. Project 1과 거의 동일하나, learning_rate를 0.1만 사용한다.
- 마찬가지로 gaussian distribution을 통해 Weights를 initialization한다.
- Activation function은 sigmoid function을 사용하였다.

```
    # train the neural network #
    def train(self, input_array, target_array):
        # convert list to matrix: [1, 2, 3, 4] to [[1], [2], [3], [4]].T
        inputs = input_array
        targets = target_array

        ## Forward Propagation ##

        # input to hidden layer
        hidden_inputs = np.dot(self.W_ih, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)

        # hidden to output layer
        final_inputs = np.dot(self.W_ho, hidden_outputs)
        final_outputs = self.activation_function(final_inputs)

        ## Backward Propagation ##
        # error
        output_errors = targets - final_outputs

        # hidden_errors are distributed according to the proportion of weights from output_errors
        # ignore the denominator for simplification
        hidden_errors = np.dot(self.W_ho.T, output_errors)

        # update the weights
        self.W_ho += self.lr * np.dot((output_errors * final_outputs * (1.0 - final_outputs)), np.transpose(hidden_outputs))
        self.W_ih += self.lr * np.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), np.transpose(inputs))
```

- Project 1과 마찬가지로, train member function에서 Weights를 update한다. Backward propagation이 사용되었다.

```
    # query the neural network #
    def query(self, img_array):
        inputs = img_array

        ## Forward Propagation ##

        # input to hidden layer
        hidden_inputs = np.dot(self.W_ih, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)

        # hidden to output layer
        final_inputs = np.dot(self.W_ho, hidden_outputs)
        final_outputs = self.activation_function(final_inputs)

        return final_outputs
```

- New query에 대해 주어진 Neural Network model에서 output을 만드는 member function이다.

```
# Initialize for the performance graph #
pts = { epoch : [ ] for epoch in epochs } # value: (angle, accuracy)
```

- 추후 each epoch에 대한 (angle, performance) graph를 그리기 위해 points를 dictionary value로 initialization 하였다.

```
## train & test the neural network ##
for epoch in epochs: # 5, 10
    for angle in angles:
        # initialization new instance for each angle
        instance = Neural_Network() # input_size = 784, hidden_size = 200, output_size = 10, learning_rate = 0.1
        ### train the neural network ###
        for _ in range(epoch):
            for img_array, label_array in augmented_training_data[angle]: # (img_array, label_array) of float
                instance.train(img_array, label_array) # present instance's weights update

        ### test the neural network ###
        # scorecard for how well the network performs, initially empty
        scorecard = []
        for img_array, label_array in testing_data: # (img_array, label_array) of float
            output_array = instance.query(img_array) # using updated weights

            # the index of the highest value corresponds to the label
            output = np.argmax(output_array)
            target = np.argmax(label_array)
            # append correct or incorrect to list
            if (output == target):
                # network's answer matches correct answer, add 1 to scorecard
                scorecard.append(1)
            else:
                # network's answer doesn't match correct answer, add 0 to scorecard
                scorecard.append(0)

        # calculate the performance score, the fraction of correct answers
        scorecard_array = np.array(scorecard, dtype=np.float64)
        # performance score
        accuracy = scorecard_array.sum() / scorecard_array.size
        # print accuracy
        print(f"Performance: {accuracy} | angle: {angle}, epoch: {epoch}")
        # store coordinate data for result graph #
        pts[epoch].append( (angle, accuracy) )
```

- Epoch가 주어질 때, each angle에 대하여 instance를 initialization하고, training과 testing을 통해 performance를 계산한다. Project 1과 같은 방식이 사용되었다.

```
## visualizing the results with graph: including marker and label ##
x = { epoch : [ angle for angle, accuracy in pts[epoch] ] for epoch in epochs }
y = { epoch : [ accuracy for angle, accuracy in pts[epoch] ] for epoch in epochs }

## plot the graph together ##
plt.plot( x[epochs[0]], y[epochs[0]], 'bD-', label = f'{epochs[0]} epochs' ) # epoch = 5
plt.plot( x[epochs[1]], y[epochs[1]], 'rD-', label = f'{epochs[1]} epochs' ) # epoch = 10

# title of the graph
plt.title("Performance and Epoch\nMNIST dataset with 3-layer neural network")
# title of each axis
plt.xlabel("additional images at a+/- angle (degrees)")
plt.ylabel("performance")
# label of each graph
plt.legend()
# grid of y-axis
plt.grid(axis = 'y')

plt.show()
```

- 마지막으로, graph를 그리기 위해 앞에서의 pts dictionary를 사용한다. Project 1과 같은 방식이 사용되었다.

Print_Result

```
C:\Users\HOME\AppData\Local\Programs\Python\Python313\python.exe C:\Users\HOME\Desktop\VS_Code\2024_MECH_Intro_AI\Project_2\Project_2.py
StudentID: 202011101    Name: 서정호
angle = 0: before training dataset augmentation for comparison.
Performance: 0.9737 | angle: 0, epoch: 5
Performance: 0.9722 | angle: 5, epoch: 5
Performance: 0.9715 | angle: 10, epoch: 5
Performance: 0.9652 | angle: 15, epoch: 5
Performance: 0.934 | angle: 20, epoch: 5
Performance: 0.9075 | angle: 25, epoch: 5
Performance: 0.9772 | angle: 0, epoch: 10
Performance: 0.9733 | angle: 5, epoch: 10
Performance: 0.969 | angle: 10, epoch: 10
Performance: 0.9574 | angle: 15, epoch: 10
Performance: 0.926 | angle: 20, epoch: 10
Performance: 0.9062 | angle: 25, epoch: 10

Process finished with exit code 0
```

StudentID: 202011101 Name: 서정호

angle = 0: before training dataset augmentation for comparison.

Performance: 0.9737 | angle: 0, epoch: 5

Performance: 0.9722 | angle: 5, epoch: 5

Performance: 0.9715 | angle: 10, epoch: 5

Performance: 0.9652 | angle: 15, epoch: 5

Performance: 0.934 | angle: 20, epoch: 5

Performance: 0.9075 | angle: 25, epoch: 5

Performance: 0.9772 | angle: 0, epoch: 10

Performance: 0.9733 | angle: 5, epoch: 10

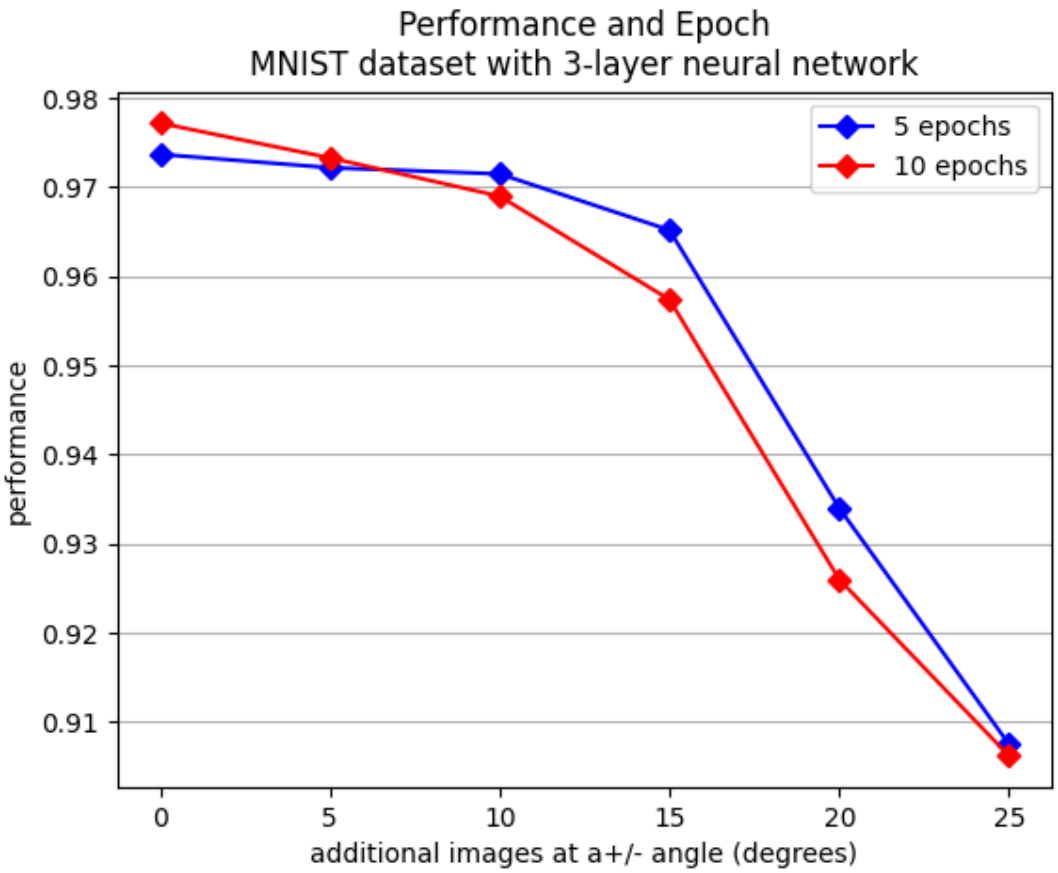
Performance: 0.969 | angle: 10, epoch: 10

Performance: 0.9574 | angle: 15, epoch: 10

Performance: 0.926 | angle: 20, epoch: 10

Performance: 0.9062 | angle: 25, epoch: 10

Graph



- angle = 0 Vs angle = 5: training_data의 augmentation 전, 후를 비교할 수 있다. Every epoch에 대해 performance가 감소했음을 알 수 있다.
- 일반적으로 augmentation은 training_data의 양을 늘려서 model의 performance를 도모한다. 그러나 **rotation을 할 때 새로 생긴 여백에 cval value로 padding 하였으므로, image가 변형되면서 오히려 성능이 저하**되었다.
- 이는 every epoch에서 angle이 커질수록 performance가 감소하는 것에서 확인할 수 있다. Rotation angle가 커질 때마다 cval값으로 padding되는 pixels이 많아졌기 때문에, image가 더 심하게 변형되어(**noise**) 되려 성능이 감소하였다.
- 따라서 data augmentation을 통해 성능을 향상시키려면, **image에 심한 변형을 가하지 않는 방식으로 augmentation**을 해야만 한다. Image를 수평 혹은 수직으로 뒤집거나(**Flip**), 일부를 잘라서 부분적인 정보에도 robust하게 만들거나(**Crop**), 확대/축소하여 크기에 대하여 generalization을 향상시키거나(**Scale**), 밝기와 채도 등을 변화시킴(**Color Jitter**)으로서 **noise**를 줄이며 augmentation을 할 수 있다.
- 한편, 10 epochs의 경우 5 epochs에 비해 성능 변화가 가파르므로, overfitting을 의심해볼 수 있다.