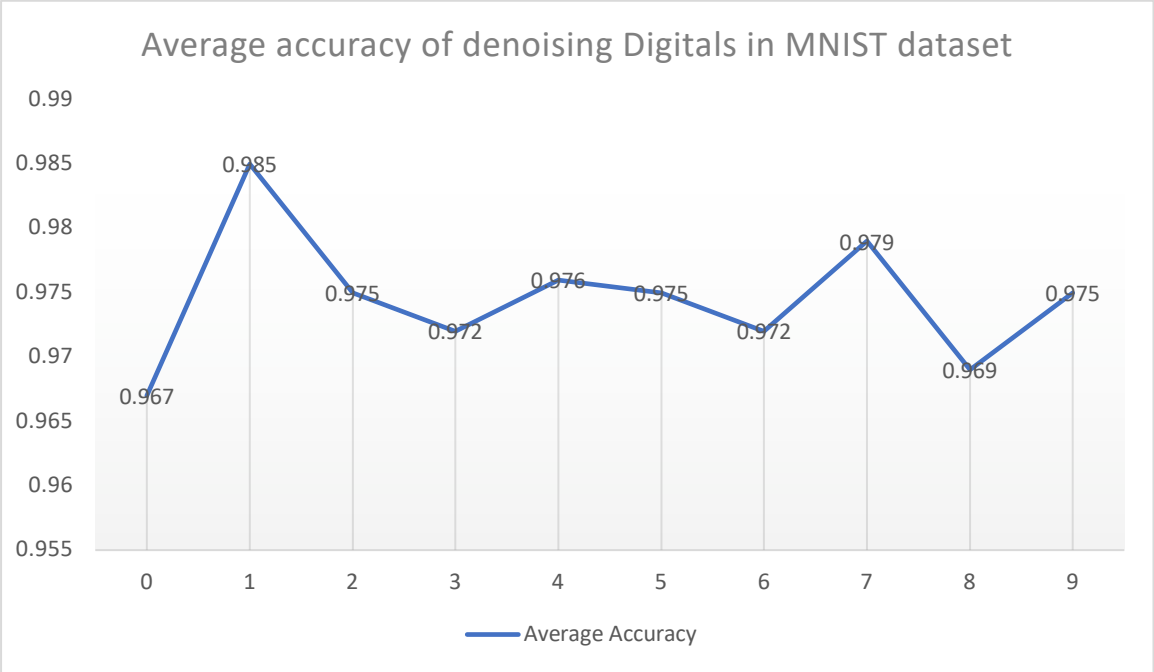
CS498: AML HOMEWORK 9

Hao Wu (haow11), teammate Shuyue Lai (shuyuel2), Dec. 3, 2018

1. Average accuracy on the first 500 images

(Table 1) Average accuracy of denoising Digitals in MINST dataset

Original Digital	0	1	2	3	4	5	6	7	8	9
Average Accuracy	0.967	0.985	0.975	0.972	0.976	0.975	0.972	0.979	0.969	0.975



(Fig 1) Average accuracy of denoising Digitals in MINST dataset

After analyzing this result, we can find it is easier to denoise easy-writing digitals, such as '1' or '7', which only have vertical line or horizontal line or both. Since the flipped points won't be likely to influence the true points.

2. Sample images (original, noisy, denoised)

Original Noise Denoised

(Fig 2) Three images of first training image with label 5

Original Noise Denoised

(Fig 3) Three images of second training image with label 0

Each image with noise has 20 random flipped points.

3. Best reconstruction (original, noisy, denoised)

Original Noise Denoised

(Fig 4) Three images of 408 training image with label 1

After constructing mean field inference, our denoised image has 0.994 of accuracy.

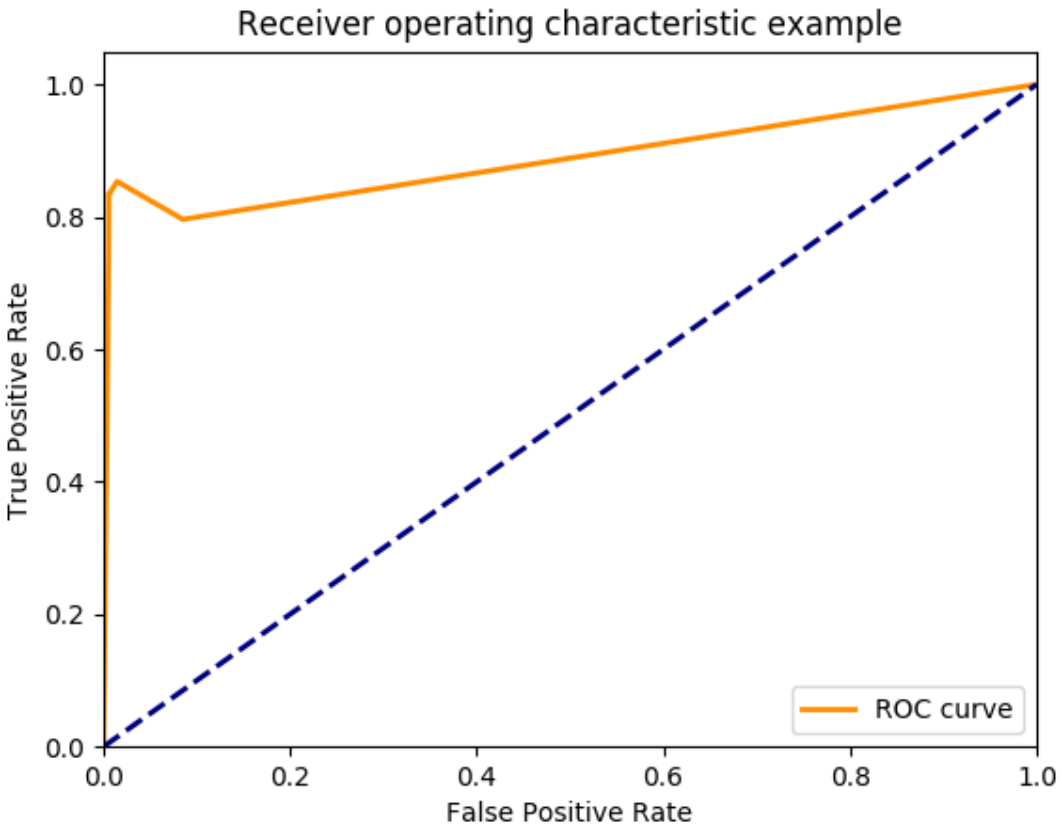
4. Worst reconstruction (original, noisy, denoised)

Original Noise Denoised

(Fig 5) Three images of 404 training image with label 8

After constructing mean field inference, our denoised image has 0.954 of accuracy.

5. ROC curve



(Fig 6) ROC of denoising flipped MINST dataset

In the experiment, we get the TPR and FPR via different c is shown below.

(Table 2) TPR and FPR of different c

c	-1	0	0.2	1	2
TPR	0.744	0.836	0.836	0.854	0.796
FPR	0.005	0.006	0.007	0.015	0.085

7. Code snippet

```
import tensorflow as tf
import numpy as np
import math
import copy
import imageio
import matplotlib.pyplot as plt

NUM = 500


# Get dataset from MINIST
def getData():
    mnist = tf.keras.datasets.mnist
    (x_train, y_train),(x_test, y_test) = mnist.load_data()
    x_train = x_train / 255.0

    for x in x_train[:NUM]:
        for i in range(len(x)):
            for j in range(len(x[0])):
                x[i][j] = 1 if x[i][j]>=0.5 else -1

    return x_train[:NUM], y_train[:NUM]


# Add noise to original dataset randomly
def addNoise(data):
    global points, rows
    data = copy.deepcopy(data)

    for i in range(len(data)):
        flippings = [np.random.randint(points) for i in range(int(points*0.02))]
        for pos in flippings:
            row, column = divmod(pos, rows)
            data[i][row][column] = -data[i][row][column]

    return data
# Update pai for MFI process
def update(row, column, pic, pai, thetaHH, thetaHX):
    global rows, columns, points
    pos = row*columns+column

    neighbors = list()
    left, up, right, down = (row, column-1), (row-1, column), (row, column+1), (row+1, column)
    if left[1]>=0: neighbors.append(pos-1)
    if up[0]>=0: neighbors.append(pos-columns)
    if right[1]<=columns-1: neighbors.append(pos+1)
    if down[0]<=rows-1: neighbors.append(pos+column)

    A, B = -pic[row][column]*2., pic[row][column]*2.
    for neighbor in neighbors:
        j_row, j_column = divmod(neighbor, columns)
        A += thetaHH*(1-2*pai[neighbor]) + thetaHX*(-1)*pic[j_row][j_column]
        B += -(thetaHH*(1-2*pai[neighbor]) + thetaHX*(-1)*pic[j_row][j_column])

    return (math.exp(A)/(math.exp(A)+math.exp(B)))



# Generate tpr and fpr for drawing ROC
def getTprFpr(predict, origin):
    predict, origin = predict.reshape((1,-1))[0], origin.reshape((1,-1))[0]
    TP, FP, FN, TN = 0, 0, 0, 0
    for i in range(len(predict)):
        if predict[i]==1 and  origin[i]==1:
            TP += 1
        if predict[i]==-1 and  origin[i]==1:
            FN += 1
        if predict[i]==1 and  origin[i]==-1:
            FP += 1
        if predict[i]==-1 and  origin[i]==-1:
            TN += 1
    return TP/(TP+FN), FP/(FP+TN)
```

fpr_list

```python
# Main MFI process
def meanFieldInference(data_flipped, thetaHH, thetaHX):
    global rows, columns, points, data, labels, accuracy_list, best_accuracy, worst_accuracy, best_pic, worst_pic
    data_recoverd = list()
    count = 0
    tpr = 0
    fpr = 0
    for pic in data_flipped:
        print('-'*20+'\nrecover picture %s'%count)
        pai = np.full((1, points),.5)[0]
        # pai: 1*784
        threshold = 1e-5
        prev_sum_pai = float('inf')
        times = 0
        while abs(prev_sum_pai-pai.sum()) >= threshold and times<10:
            print(pai.sum())
            prev_sum_pai = pai.sum()
            for i in range(points):
                row, column = divmod(i, columns)
                pai[i] = update(row, column, pic, pai, thetaHH, thetaHX)
            times += 1

        pic_recoverd = np.ones((rows,columns))

        for i in range(rows):
            for j in range(columns):
                pos = i*columns + j
                pic_recoverd[i][j] = -1 if pai[pos]>=.5 else 1

        accuracy = 1.-check(pic_recoverd==data[count])/points
        if accuracy >= best_accuracy:
            best_accuracy = accuracy
            best_pic = count
        if accuracy <= worst_accuracy:
            worst_accuracy = accuracy
            worst_pic = count
# Check the difference between denoised image and original image
def check(a):
    count = 0
    for i in range(len(a)):
        for j in range(len(a[0])):
            if not a[i][j]:
                count += 1
    return count


# Save images
def saveImages(data, name):
    for i,img in enumerate(data):
        imageio.imwrite('./result/%s_%s.jpg'%(i,name), img)


# Draw ROC
def drawPicture(tpr, fpr):
    plt.figure()
    lw = 2
    plt.plot(fpr, tpf, color='darkorange',
             lw=lw, label='ROC curve')
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

        cur_tpr, cur_fpr = getTprFpr(pic_recoverd, data[count])
        tpr = (tpr*count + cur_tpr)/(count+1)
        fpr = (fpr*count + cur_fpr)/(count+1)

        label = labels[count]
        accuracy_list[label] += accuracy
        count_list[label] += 1

        data_recoverd.append(pic_recoverd)
        count += 1

    return np.array(data_recoverd), tpr, fpr
```

```python
if __name__ == "__main__":
    thetaHHs = [.2]
    # thetaHHs = [-1, 0, 0.2, 1, 2]
    thetaHX = 2.
    accuracy_list = [0.]*10
    count_list = [0]*10
    best_accuracy, worst_accuracy = float('-inf'), float('inf')
    best_pic, worst_pic = 0, 0
    tpr_list, fpr_list = [0.], [0.]


    data, labels = getData()
    saveImages(data, 'original')
    rows, columns, points = len(data[0]), len(data[0][0]), len(data[0])*len(data[0][0])
    data_flipped = addNoise(data)
    saveImages(data_flipped, 'flipped')

    for thetaHH in thetaHHs:
        print('='*40+'\nthetaHH: %s'%thetaHH)
        data_recoverd, tpr, fpr = meanFieldInference(data_flipped, thetaHH, thetaHX)
        tpr_list.append(tpr)
        fpr_list.append(fpr)
        saveImages(data_recoverd, 'recoverd')
        average_accuracy_list = [accuracy_list[i]/count_list[i] for i in range(10)]

        for i in range(10):
            print('Num: %s\tCount: %s\tAccuracy: %s'%(i,count_list[i],average_accuracy_list[i]))

        print('Best picture: %s\tBest accuract:%s'%(best_pic, best_accuracy))
        print('Worst picture: %s\tWorst accuract:%s'%(worst_pic, worst_accuracy))

    tpr_list.append(1.)
    fpr_list.append(1.)
    tpr_list = np.array(tpr_list)
    fpr_list = np.array(fpr_list)
    print(tpr_list, fpr_list)

    # drawPicture(tpr_list, fpr_list)
```