2 Practical Review on Hopfield

Below is the code to train a hopfield network to store a small dataset and then recover them from a noisy dataset.

Before feeding any photo to the network we resize it to be 40x40, So the network has 1600 neurons and the weights are stored in a 1600x1600 matrix. The rest of the code is exactly the same as previous assignment.

The networks outputs from noisy inputs is evaluated both pixel-by-pixel and qualitatively.

[ ]:
```
import numpy as np
from PIL import Image
import os

dim = 40
network_dim = dim ** 2


class Hopfield:
    def __init__(self):
        self.weight_matrix = np.zeros((network_dim, network_dim))

    def update_weights(self, input):
        for i in range(len(input)):
            for j in range(len(input)):
                if (i == j):
                    self.weight_matrix[i, j] = -50
                    break
                self.weight_matrix[i, j] += input[i] * input[j]
                self.weight_matrix[j, i] = self.weight_matrix[i, j]

    def train(self, dir):
        for photo_path in dir:
            image_matrix, image_size = read_image(photo_path, 'images/')
            image_matrix = image_matrix.flatten()
            self.update_weights(image_matrix)

    def restore(self, input):
        output = []
        for row in range(len(self.weight_matrix)):
            temp = 0
            for column in range(len(self.weight_matrix)):
                if row == column:
                    continue
                temp += self.weight_matrix[row, column] * input[column]
            output.append(sign(temp))
        return np.array(output, dtype='uint8')
```

```
    def feed(self, dir):
        for photo_path in dir:
            image_matrix, image_size = read_image(photo_path, 'noisy_images/')
            image_matrix = image_matrix.flatten()
            output = self.restore(image_matrix)
            output = np.reshape(output, (dim, dim))
            output = Image.fromarray(output, mode='L').resize(image_size)
            output.save('results/' + photo_path)


def sign(a):
    if a >= 0:
        return 255
    else:
        return 0


def read_image(path, folder):
    img = Image.open(folder + path).convert(mode="L")
    size = img.size
    img = img.resize((dim, dim))
    imgArray = np.asarray(img, dtype=np.uint8)
    x = np.zeros(imgArray.shape, dtype=np.float)
    x[imgArray > 60] = 1
    x[x == 0] = -1
    return x, size


def evaluate_outputs():
    return 1


if __name__ == '__main__':
    hopfield = Hopfield()
    dir = os.listdir('images')
    dir.remove('.DS_Store')
    hopfield.train(dir)

    dir = os.listdir('noisy_images')
    dir.remove('.DS_Store')
    hopfield.feed(dir)
```

[6]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import os
```

```
import re

errors_10_16 = []
errors_30_16 = []
errors_60_16 = []

errors_10_32 = []
errors_30_32 = []
errors_60_32 = []

errors_10_64 = []
errors_30_64 = []
errors_60_64 = []


def draw_table():
    table_data = []

    table_data.append(['16', str(round(np.average(errors_10_16), 2)) + ' %',
                       str(round(np.average(errors_30_16), 2)) + ' %',
                       str(round(np.average(errors_60_16), 2)) + ' %'])
    table_data.append(['32', str(round(np.average(errors_10_32), 2)) + ' %',
                       str(round(np.average(errors_30_32), 2)) + ' %',
                       str(round(np.average(errors_60_32), 2)) + ' %'])
    table_data.append(['64', str(round(np.average(errors_10_64), 2)) + ' %',
                       str(round(np.average(errors_30_64), 2)) + ' %',
                       str(round(np.average(errors_60_64), 2)) + ' %'])

    fig, ax = plt.subplots()
    table = ax.table(cellText=table_data, cellLoc='center'
                     , colLabels=['Font Size', '10% Noise', '30% Noise', '60%␣
 ↪Noise'])
    table.set_fontsize(15)
    table.scale(3, 3)
    fig.canvas.draw()
    bbox = table.get_window_extent(fig.canvas.get_renderer())
    bbox = bbox.from_extents(bbox.xmin - 5, bbox.ymin - 5, bbox.xmax + 5, bbox.
 ↪ymax + 5)
    bbox_inches = bbox.transformed(fig.dpi_scale_trans.inverted())
    ax.axis('off')
    fig.savefig('error_table.png', bbox_inches=bbox_inches)
    plt.show()


def read_image_dir(folder, name, size):
    path = name + size + '.bmp'
    img = Image.open(folder + path).convert(mode="L")
```

```
    imgArray = np.asarray(img, dtype=np.uint8).flatten()
    x = np.zeros(imgArray.shape, dtype=np.float)
    x[imgArray > 60] = 1
    x[x == 0] = 0
    return x


def read_image_path(path, folder):
    img = Image.open(folder + path).convert(mode="L")
    imgArray = np.asarray(img, dtype=np.uint8).flatten()
    x = np.zeros(imgArray.shape, dtype=np.float)
    x[imgArray > 60] = 1
    x[x == 0] = 0
    return x


def add_to_array(size, error, data):
    if error == '10':
        if size == '16':
            errors_10_16.append(data)
        elif size == '32':
            errors_10_32.append(data)
        elif size == '64':
            errors_10_64.append(data)
    elif error == '30':
        if size == '16':
            errors_30_16.append(data)
        elif size == '32':
            errors_30_32.append(data)
        elif size == '64':
            errors_30_64.append(data)
    elif error == '60':
        if size == '16':
            errors_60_16.append(data)
        elif size == '32':
            errors_60_32.append(data)
        elif size == '64':
            errors_60_64.append(data)


if __name__ == '__main__':
    dir = os.listdir('images')
    dir.remove('.DS_Store')

    dir_noise = os.listdir('results')
    dir_noise.remove('.DS_Store')
```

```
    for photo in dir_noise:
        error, name = re.split('_|16|32|64|.bmp', photo)[0:2]
        size = re.split('_|A|B|C|D|E|F|G|H|I|J|.bmp', photo)[2]

        img = read_image_dir('images/', name, size)
        img_noise = read_image_path(path=photo, folder='noisy_images/')
        add_to_array(size, error, 100 * (1 - np.average(np.abs(img -␣
↪img_noise))))

    draw_table()
```

Font Size	10% Noise	30% Noise	60% Noise
16	90.47 %	68.95 %	53.31 %
32	89.83 %	70.06 %	50.78 %
64	90.34 %	70.13 %	52.71 %

Below is the accuracy table which is calculated qualitatively:

Font Size	10% Noise	30% Noise	60% Noise
16	70 %	50 %	10 %
32	80 %	70 %	0 %
64	80 %	50 %	10 %